NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

BUILDING A DYNAMIC WEB/DATABASE INTERFACE

by

Julie Cornell

December, 1996

Thesis Advisor: C. Thomas Wu

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE TITLE OF THESIS. BUILDING A DYNAMIC WEB/DATABASE INTERFACE	5. FUNDING NUMBERS
6. AUTHOR(S) Cornell, Julie L.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

This thesis examines methods for accessing information stored in a relational database from a Web Page. The stateless and connectionless nature of the Web's Hypertext Transport Protocol as well as the open nature of the Internet Protocol pose problems in the areas of database concurrency, security, speed, and performance.

We examined the Common Gateway Interface, Server API, Oracle's Web/database architecture, and the Java Database Connectivity interface in terms of performance and flexibility. Oracle's approach was found to be the most robust and best performing approach currently in use, although the Java Database Connectivity interface has not yet been widely implemented.

Based on our research and experience implementing a prototype, we conclude that Web/database technology is currently only appropriate for read-only type applications such as Decision Support Systems and Information Delivery Systems. The database access methods presently available cannot support more advanced capabilities of client/server type applications including client-side data validation, sophisticated user interfaces, and concurrency among multiple users.

14. SUBJECT TERMS World Wide Web, Database, Internet, Security, Concurrency, Performance, Client/Server			15. NUMBER OF PAGES 125
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

Approved for public release; distribution is unlimited.

BUILDING A DYNAMIC WEB/DATABASE INTERFACE

Julie Cornell
B.S., Santa Clara University, June 1991

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1996

Author: Julie Cornell
Julie Cornell

Approved by:
C. Thomas Wu, Thesis Advisor

Deborah Paquette Davis
Deborah Paquette Davis, Second Reader

Ted Lewis
Theodore Lewis, Chairman
Department of Computer Science

ABSTRACT

This thesis examines methods for accessing information stored in a relational database from a Web Page. The stateless and connectionless nature of the Web's Hypertext Transport Protocol as well as the open nature of the Internet Protocol pose problems in the areas of database concurrency, security, speed, and performance.

We examined the Common Gateway Interface, Server API, Oracle's Web/database architecture, and the Java Database Connectivity interface in terms of performance and flexibility. Oracle's approach was found to be the most robust and best performing approach currently in use, although the Java Database Connectivity interface has not yet been widely implemented.

Based on our research and experience implementing a prototype, we conclude that Web/database technology is currently only appropriate for read-only type applications such as Decision Support Systems and Information Delivery Systems. The database access methods presently available cannot support more advanced capabilities of client/server type applications including client-side data validation, sophisticated user interfaces, and concurrency among multiple users.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

API	Application Program Interface
CGI	Common Gateway Interface
DBMS	Database Management System
DCD	Database Connection Descriptor
DFAS	Defense Finance and Accounting Service
DLL	Dynamic Link Library
DMDC	Defense Manpower Data Center
DTIC	Defense Technical Information Center
FTP	File Transfer Protocol
GIF	Graphics Interchange Format
HTML	Hypertext Markup Language
HTF	Hypertext Function
HTP	Hypertext Procedure
HTTP	Hypertext Transport Protocol
IBM	International Business Machines
IP	Internet Protocol
ISAPI	Internet Server Application Program Interface
JDBC	Java Database Connectivity
JFTR	Joint Federal Travel Regulation
LAN	Local Area Network
MIHA	Moving-In Housing Allowance
MIME	Multipurpose Internet Mail Extension
NC	Network Computer
NCSA	National Center for Supercomputing Applications
NSAPI	Netscape Server Application Program Interface
ODBC	Open Database Connectivity
OHA	Overseas Housing Allowance

OWA	Oracle Web Agent
PC	Personal Computer
PDTATAC	Per Diem, Travel, and Transportation Allowance Committee
PL/SQL	Procedural Language / Structured Query Language
SQL	Structured Query Language
SSL	Secure Sockets Layer
URL	Uniform Resource Locator
VPOS	Virtual Point of Sale
WRB	Web Request Broker
WRBX	Web Request Broker Execution Engine
WWW	World Wide Web

I. INTRODUCTION

A. BACKGROUND

During the past several years, the World Wide Web has become increasingly popular as a means of displaying and accessing information. The Web technology has the potential for a tremendous impact on many aspects of our everyday lives, but currently, the technology is still rushing to catch up with the expectations of users. Although the Web has only been in existence for a relatively brief period of time, it has already undergone a rapid process of evolution.

Initially, Web pages were used simply to display static information. Hypertext Markup Language (HTML) files containing formatted text and images were stored on a Web server, where they could then be viewed by users all over the world. This was an extremely desirable capability, as the same data could transparently be presented to users in various locations, running on various platforms, various networks, and using various browsers. However, large Websites rapidly became unmanageable. The maintenance of hundreds and even thousands of static files became a monumental task, given the volatile nature of most data. Any change in the format or presentation of the Web page would often have to be made by hand in multiple files. Furthermore, these static Web pages tended to be lacking in content. Users often desired the ability to obtain more detailed information which was not made available because of the difficulty of maintaining it.

These factors led to a second stage of evolution for the Web, in which HTML files were generated by a program. In many cases, the information that companies wanted to present on the Web was already stored in relational databases. Instead of extracting information from the database and formatting it by hand, people began to write programs to automate this process. This prevented a great deal of tedious HTML formatting, and made maintenance much simpler, as any changes in the presentation or content of the information would only have to be made in the program. An additional benefit was that each of the pages would be consistent with one another. However, the HTML files still

had to be stored and maintained on the Web server, and the information had to be updated from the database frequently in order to stay current.

Because of the problems of generating and maintaining static HTML files, the World Wide Web is currently entering a third stage of evolution. In this stage, Web pages will be dynamically generated upon request, pulling data directly from a database. The advantages of this approach are clear. Multiple files will no longer have to be stored on the Web server. Changes in format will not have to be propagated throughout multiple files. Also, the data presented will always be up to date and accurate, as it will come directly from the database. The union of Web and database technologies is the next logical step in this evolution process.

B. OBJECTIVES AND RESEARCH QUESTIONS

The purpose of this study is to examine and analyze methods for accessing information stored in a relational database from a Web Page. In this thesis, I will discuss some of the different alternatives for approaching this problem and summarize the advantages and disadvantages of each approach. I will also explore various issues involved in building a Web/database interface.

The primary research question to be addressed in this thesis is:

- What methods can be used to access a database from a Web Page and what are the issues involved?

The subsidiary research questions are:

- Why would one want to incorporate information from a database into a Web Page?
- What are the basic characteristics of Web/database applications and how do they differ from traditional client/server database applications?
- What are the advantages and disadvantages of the various approaches?
- How will speed and performance issues be handled?

- What concurrency problems are associated with this method of database access?
- How will security concerns be addressed?
- What will be the future of this technology?

C. SCOPE

This study will focus primarily on the examination and evaluation of different approaches to solving the problem of integrating database information with the World Wide Web. Although there are many database design and normalization issues that are related to this topic, I will exclude these issues and assume that a relational database already exists that one would like to access via the Web. In addition, issues relating to Web server setup and administration will also be excluded.

Furthermore, in the area of security, I will focus on the issues that relate to the security of the database itself and not cover the other security issues that surround Web development.

D. METHODOLOGY

This thesis will be written in conjunction with a project that consists of taking an actual existing database and creating a prototype for a Web Page that will dynamically display information from the database based on user input. The database to be accessed is the Overseas Housing Allowance (OHA) database, which is being developed by the Defense Manpower Data Center (DMDC) on behalf of the Per Diem, Travel and Transportation Allowance Committee (PDTATAC). I will incorporate the knowledge that I gain from working on this project with research of current literature and product documentation to complete the study.

E. MOTIVATION

The integration of Web and Database technologies is an area that has the potential to revolutionize the way that we use computers and access information, and in general, change the way business is conducted. There is a great demand for the capability to access database information through the Web. Many products are now beginning to come onto the market which claim to easily link a database to a Web application, and in addition, most major database vendors have begun to deploy their own software to make their databases accessible to the Web. Because of the rapid evolution of the Web, it is an extremely volatile environment, with new products and developments occurring almost daily. As a result, the field of Web development is surrounded by a great deal of hype and misinformation. Despite the potential of this field, there are many dangers of rushing into a technology that has not yet reached maturity. The urgency to develop this capability may lead to products and procedures that are not well designed and have not taken all of the issues into consideration. It is worthwhile to carefully consider all of the issues involved and develop sound methods for connecting a database to a Web Page.

F. ORGANIZATION OF STUDY

This thesis is organized as follows: Chapter II contains a discussion on the basic characteristics of Web/database applications and how they are fundamentally different from traditional client/server database applications. The advantages and disadvantages of the Web approach are also discussed. Chapter III provides an overview of the different approaches that are currently available, how they work, the advantages and disadvantages of each, and particular issues relating to each approach. Chapter IV contains discussions of speed and performance, concurrency, and security issues relating to Web/database development, and how different types of database applications are affected by these issues. In Chapter V, a summary of the OHA Web/database case study is given. This includes background information of the project, the tools and methods used to complete

the project, and specific issues and problems that were encountered. Finally, Chapter VI contains speculation on the future of this technology and general conclusions.

The appendices contain the database definition, program code, generated HTML, and screen views for the OHA Web/database interface.

II. THE FUNDAMENTAL CHARACTERISTICS OF WEB/DATABASE APPLICATIONS

A. BACKGROUND

Currently, the most common method for users to access information in a database is with a client/server database application, or *front end.* This is a program, usually supporting a Graphical User Interface (GUI), that presents an easy-to-use interface to the user, thus restricting the actions the user is able to perform and circumventing the need for the casual database user to learn complex database manipulation languages such as Structured Query Language (SQL). In many cases, these types of applications could be replaced with a Web/database application. There are substantial advantages to the Web approach, but there are also serious obstacles that must be overcome before the Web technology can completely replace the need for client/server applications.

B. ARCHITECTURE

Client/server applications and Web/database interfaces may appear essentially the same to the end user, but there are some very fundamental differences in how they work that have an important impact on their functionality. In a client/server application, the application program runs on the end user's computer (the client) and communicates with the database (the server) through a network or modem connection. The application passes SQL statements through an Application Program Interface (API) to the database and the results are returned to the client machine and displayed to the user. This scenario is depicted in Figure 1.

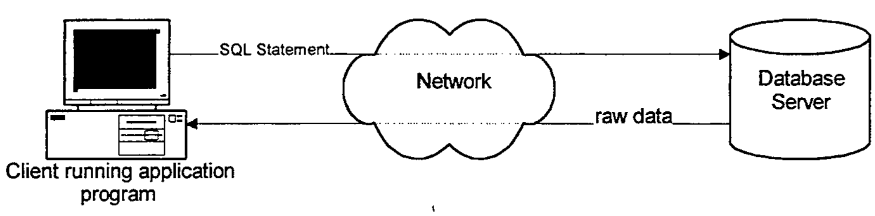


Figure 1. Client/Server Architecture

Web/database interfaces operate differently from client/server applications. Although there are several ways in which a database can be accessed through the Web, they generally have certain characteristics in common. Like client/server applications, Web applications also display information in a GUI. However in a Web interface, the GUI is provided by the Web browser, which runs on the client machine, as opposed to the GUI being provided by the particular programming language used in a client/server application. In this scenario, the Web server provides an additional layer between the client and the database server. The user specifies a Uniform Resource Locator (URL), which uniquely identifies a particular Web server to connect to and an HTML file to view or a program to run on that server. In order to access the database, a program must be specified. The Web server interprets the URL and dispatches the program, which can then access the database in a variety of ways (which will be discussed in Chapter III). The program formats the retrieved data into HTML, which is then returned to the client machine to be displayed in the browser. See Figure 2 for a diagram of this configuration.

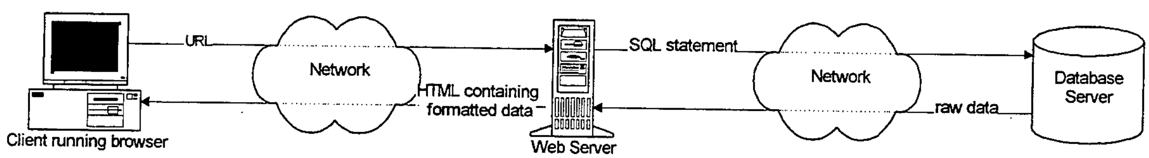


Figure 2. Web/Database Interface Architecture

C. ADVANTAGES OF THE WEB APPROACH

As many application programmers, project managers, and other information systems professionals have learned over the past several years, client/server database applications are inherently difficult to build, deploy, and manage. Billions of dollars have been invested in client/server technology, and many tools and products have been developed in support of client/server applications, but it remains an extremely

problematic field. The characteristics of Web technology make it possible to circumvent many of the problems associated with client/server development.

One of the biggest problems that plagues client/server applications is that of deployment and maintenance. The application program executables, along with various other system files that are necessary to run it, must be installed on each end user's machine. This usually requires that a setup or installation package be written, which is then distributed to each of the users. It is often the case that different end users have different system configurations, which causes conflicts with the client/server software being installed and makes it necessary for the organization deploying the application to provide extensive technical support to its end users. In addition, if a system is being developed for use by end users running on different platforms, the application must be ported to each of the intended platforms. Once the application has been deployed, any software changes require that the system be re-installed on each end user's machine. All of these factors make the administration of a client/server application a very tedious and cumbersome task.

By using a Web application, all of these problems are avoided. The programs to retrieve and manipulate data are stored in one place: either on the Web server or in the database itself. This eliminates the need to install software on each end user's machine and makes software changes much easier. Furthermore, users running on different platforms can use the same programs. There is no need to write different versions for different platforms, as Web browsers are available for a wide variety of platforms and HTML is machine independent. These advantages make Web technology a very desirable alternative to client/server development.

Another problem that client/server developers face is that of connectivity. Client/server applications have to deal with users connecting by modems, Internet connections, and local network connections as well. Most commercial Database Management Systems (DBMS) have their own network software, which requires expensive site licenses for each user. Web/database applications do not have this problem. The client simply needs to have Internet or modem access to the Web, and the application developers need only worry about the connectivity between the Web server

and the database. Furthermore, because Web applications do not maintain a persistent connection to the database, the database server can handle more users at a time.

One final benefit of the Web approach is that all users, no matter what their level of sophistication, are generally very comfortable using a Web browser. This is part of the appeal of the World Wide Web that has made it so popular. The fact that many people are already familiar with the use of a browser eases part of the learning curve associated with the deployment of any application. This will make it easier to train end users on the use of the system being developed. Like any application, a poorly written Web application will be more difficult to use than a well written one, but the standardized features of a Web browser provide users with a familiar framework.

D. DISADVANTAGES OF THE WEB APPROACH

While the advantages of the Web approach over client/server development may seem overwhelming, there remain many obstacles for Web technology to overcome before it can completely replace client/server technology.

Probably the biggest obstacle that Web developers have encountered is the *stateless* nature of the Hypertext Transport Protocol (HTTP) server. Whereas client/server applications have a program running throughout the various screens of the application, Web applications consist of completely independent pages and have no persistent program memory to save user state information. Therefore, it is necessary to store user state information on the client side. As a simple example of why this is a problem, suppose a corporate application requires the user to select from a list of customers on the initial screen in order to display various information about that customer on the next screen. In a client/server application, the customer name and possibly an identifier would simply be stored as variables in memory and that information would be accessible in subsequent screens. In a Web application, however, the information must be explicitly passed on to the next page, as each Web page has no knowledge of previous pages. In this example, there is only a small amount of state information to be passed to the next page, but even in a relatively simple Web application the amount of data that

must be stored can quickly escalate to unmanageable levels. There are several methods currently being employed by developers as workarounds to this problem.

1. Embedding State Information in HTML

One method for passing state information that is commonly used is to embed the information directly in the HTML. HTML provides mechanisms for passing name-value pairs of information from one Web page to the next. These name-value pairs are passed along with the URL in the following format: DATA=123, where DATA is the name of the identifier and 123 is the value that is to be passed. There are several ways to take advantage of this feature, which are illustrated the examples in Figures 3 and 4. Figure 3 contains the actual HTML for the examples, and Figure 4 shows how these examples would actually look to the user when displayed in a browser.

In Example 1, there are three links, each of which have name-value pairs embedded directly into the URL. If the user clicks on the "Apples" link, the program prog1.pl (in this case, a Perl program), will be invoked, with a value of 1 for argument DATA1. Each of the links call the same program, but have different arguments, thus causing different behavior.

By using form tags, a Web page can capture information that the user enters on the screen and pass it as parameters to another program. There are a variety of form controls available in HTML, including check boxes, list boxes, text boxes, and radio buttons. Each of the form controls has a NAME attribute and a VALUE attribute, which constitute the name-value pairs. Example 2 shows a text control. In the case of a text control, the VALUE attribute is the text that the user types in the text box. The actual program to be executed is indicated by the ACTION attribute of the FORM tag. When the user hits the SUBMIT button, the text that has been entered in the text box will be passed as argument TEXT1 to the program prog2.pl. Example 3 demonstrates the use of radio buttons. These operate in a similar manner. When the user clicks on one of the radio buttons and hits the SUBMIT button, the corresponding value is passed to prog2.pl as parameter RADIO1.

If some information needs to be passed throughout several screens, then hidden fields can be used, as in Example 4. These fields can not be seen by the user, but they can contain information that will be needed in subsequent screens. In this case, this hidden field will generate the name-value pair "DATA3=HELLO".

These methods are useful for passing small amounts of data between screens, but for larger and more complex applications, other methods become necessary.

```
<HTML>
<HEAD>
<TITLE>Examples</TITLE>
</HEAD>
<BODY>

<!--Example 1-->
<H4>Embedding state information in the URL</H4>
<A HREF="http://www.whatever.com/cgi-bin/prog1.pl?DATA1=1">Apples</A>
<BR>
<A HREF="http://www.whatever.com/cgi-bin/prog1.pl?DATA1=2">Oranges</A>
<BR>
<A HREF="http://www.whatever.com/cgi-bin/prog1.pl?DATA1=3">Pears</A>
<BR>
<HR>

<H4>Using a form to create name-value pairs</H4>
<FORM METHOD="POST" ACTION="/cgi-bin/prog2.pl">

<!--Example 2: Using a text control to capture information entered-->
<!--by the user-->
Enter data here: <INPUT TYPE="text" NAME="TEXT1">
<BR>

<!--Example 3: Using a radio buttons to capture information entered-->
<!--by the user-->
<INPUT TYPE="radio" NAME="RADIO1" VALUE="CA" CHECKED> California
<BR>
<INPUT TYPE="radio" NAME="RADIO1" VALUE="OR"> Oregon
<BR>
<INPUT TYPE="radio" NAME="RADIO1" VALUE="WA"> Washington
<BR>

<!--Example 4: Using a hidden field to pass name-value pairs-->
<INPUT TYPE="hidden" NAME="DATA3" VALUE="HELLO">
<!--When the Submit button is hit, the CGI program in this example-->
<!--is invoked with the name-value pairs from the form as arguements-->
<INPUT TYPE="submit" VALUE="SUBMIT">
</FORM>
</BODY>
</HTML>
```

Figure 3. Examples of Embedding State Information in HTML

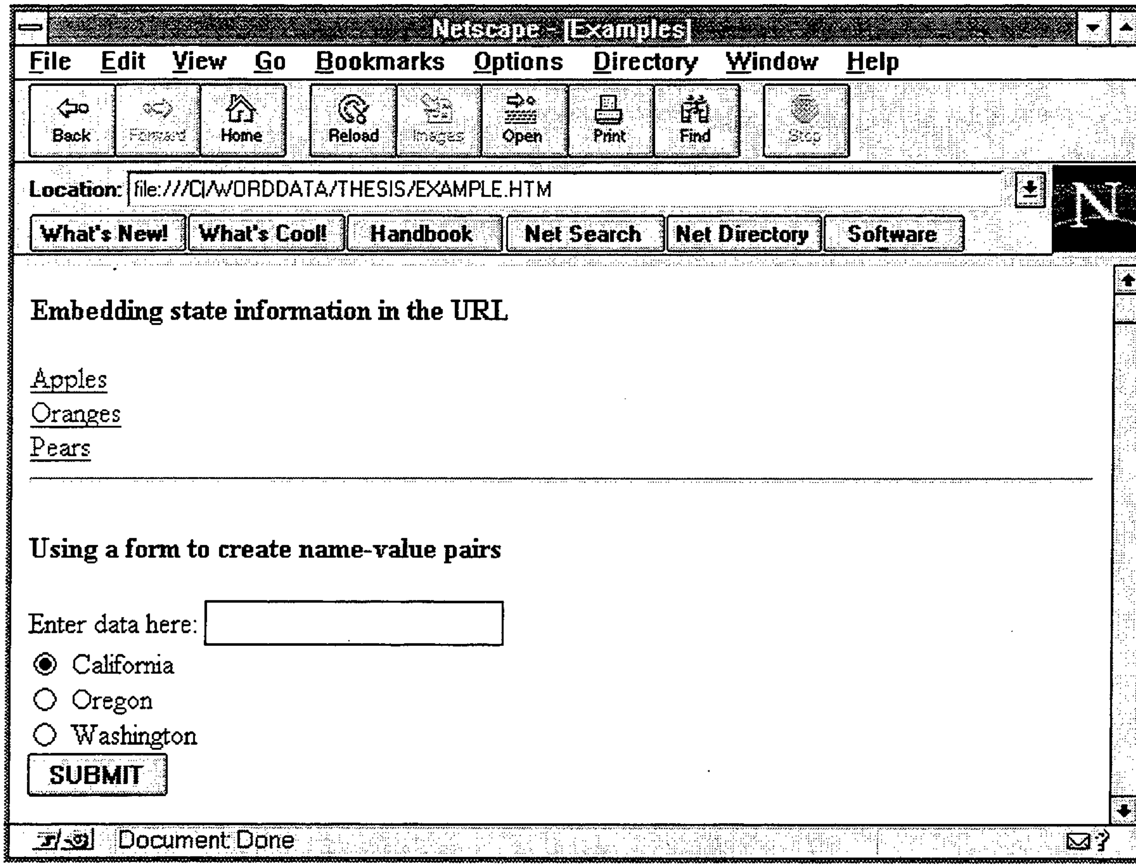


Figure 4. HTML with Embedded State Information as Displayed in a Browser

2. HTTP Cookies

Another way to save state information on the client side is by the use of HTTP Cookies. A cookie is simply a small text file that is stored on the user's computer. A Web/database application can write state information to a Cookie in the form of name-value pairs, and then retrieve it on another page. Cookies can also have expiration dates, so that the state information will expire between sessions.

This approach can be useful when the amount of information to be stored becomes too large and cumbersome to be passed around in hidden fields or when an application needs to save state information between user sessions. However, one disadvantage to this approach is that not all browsers support Cookies. Therefore, if this

method is used, it is important to ensure that the end users are using a browser that does support Cookies, or the application will not function properly. In addition, there are also security concerns regarding the use of cookies. It is considered unsafe to allow an untrusted application to write information to one's hard disk over the Internet for fear that the information may somehow contain a virus.

3. Saving State Information in the Database

One final possibility for solving the statelessness problem is to simply write information back to the database and then retrieve it in subsequent pages. This method could be used in instances where the amount of data is too large for either of the preceding methods. However, this approach requires a great deal of planning in the early stages of the application development, as the database design must take into account the types of information that will need to be stored.

Perhaps future versions of the HTTP protocol will solve the problems caused by statelessness, but in the current state of Web technology, it is up to the programmer to maintain the user's state.

Another problem with Web/database development that follows from the stateless nature of the Web is that Web/database applications are also *connectionless*. In a client/server application, the user connects to the database and remains connected throughout the session. Web applications, however, must establish a new database connection with each new page. This problem has performance and concurrency ramifications which will be discussed in Chapter IV.

A third difficulty with Web/database interfaces is that the programmer has less control over the user's environment. While a client/server application developer has a great deal of control over the appearance and execution of the application, the appearance of a Web application is determined by the particular browser and platform that the end user is running. The same characteristics that provide greater flexibility and portability in Web applications also lead to a loss in control in the way the application looks and behaves. A Web page that looks nice on the developer's screen may have a very

different appearance to an end user. This is a tradeoff which must be considered in deciding whether to choose the Web approach over client/server technology. Any Web application should be viewed on a variety of browsers and platforms before being deployed.

In addition to the lack of control over appearance, the Web developer also has less control over the flow of execution than in a client/server environment. The navigation capabilities of browsers allow users to go back to the previous screen at any time, start in the middle of the application, or go halfway through the application and then quit. The programmer must take all of these possibilities into consideration.

There are also significant drawbacks to the Web approach in the area of security. Issues relating to Web/database security will be discussed in detail in Chapter IV.

One final obstacle to Web/database development, and a very substantial one, is that although the Web has some powerful capabilities, the current state of Web technology can not do everything that a client/server application can do. Some of the more complex functionality of client/server applications can simply not be replicated in a Web application. For example, suppose an application design requires that the when a user clicks a particular radio button, various controls on the screen become disabled. Because a single Web page can not change dynamically once it is generated, and because there is no mechanism to enable or disable controls, this behavior can not be duplicated in the Web environment. Therefore, until Web technology progresses to include such capabilities, client/server technology will still be necessary for many applications. However, for relatively simple applications, a Web/database interface is a viable option.

III. DIFFERENT APPROACHES TO BUILDING A WEB/DATABASE INTERFACE

A. BACKGROUND

There are currently a variety of approaches available to the problem of connecting a database to the Web, and undoubtedly, better and faster ways will continue to be developed. All of the approaches require that a program be run in order to access the database. The primary factor that distinguishes one approach from another is where the program is executed. It is possible for the database access program to run on the client, on the Web server, or on the database server. Some other distinguishing characteristics of Web/database interface methods include how the program is invoked, how the program is able to access the database, and how information is returned to the client browser for display. In this chapter, I will examine examples of different ways to implement a Web/database interface including the Common Gateway Interface (CGI), Server API, Oracle's Web Request Broker/PL/SQL Agent solution, and the Java Database Connectivity (JDBC) specification.

B. THE CGI APPROACH

1. What is CGI?

CGI is a standard interface that has been developed in order to provide dynamic content in a Web page including, but not limited to, database access. CGI allows a Web page to invoke a program on the Web server and get back the results. As such, it is an example of an approach in which the database access program runs on the Web server.

CGI was the first mechanism developed to allow database access from a Web page. Hence, it is currently the most widely used and popular approach.

2. How Does CGI Work?

A CGI program can be written in a variety of languages, including C, Perl, and even Unix shell scripts. The compiled program or script resides in the directory structure of the Web server, usually in a directory called cgi-bin. HTML tags can specify a CGI program to run and parameters to be passed in, as shown in the example in Figure 3. When a user clicks on a button or a link that specifies a CGI program, the browser sends this request to the HTTP server. The HTTP server then checks to see if the requested program exists, and if the user has permission to execute it, and initiates execution of the program as a separate process. The program can then access information in a database. When the program completes execution, the results are returned to the client browser, usually in the form of pre-formatted HTML text. Figure 5 illustrates this process.

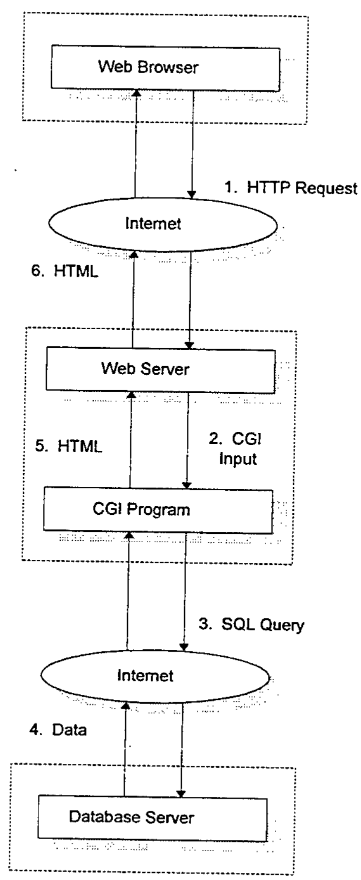


Figure 5. Web/Database Access Using CGI

3. Communication

The HTTP server and the CGI executable must be able to exchange information so that the program can get input parameters from the client and return output. This communication is facilitated by the Standard Input and Output data streams, STDIN and STDOUT. In addition, various environment variables are set by the server that contain important information that is accessible to the CGI program. This includes information about the HTTP server itself, the particular request, and the client. For example, the HTTP_ACCEPT variable contains information indicating what type of output the client's browser will accept in response to the request. This information is useful in dealing with users running a variety of browsers. If a browser is unable to accept a particular type of output, the CGI program can be set up to obtain this information from the HTTP_ACCEPT variable and reply in another format. In this way, a Web site can take advantage of the advanced features of more sophisticated browsers while still supporting older or more primitive browsers.

When a CGI program returns output to the client, it must be prefaced with a Multipurpose Internet Mail Extension (MIME) header followed by a blank line. A MIME header is simply a line of text that indicates what type of output is being returned. This tells the browser how to handle the output. Some examples of MIME headers are:

```
Content-type:  text/html
Content-type:  image/gif
```

The first example indicates a simple HTML document, and the second indicates that the output consists of a Graphics Interchange Format (GIF) image.

4. Database Access

In addition to communicating with the HTTP server, a CGI program must also be able to communicate with the database in order to perform queries. The type of database interface to be used depends upon the CGI programming language being used, the particular DBMS to be accessed, and the Web server platform. Most major DBMS's

provide an API interface that allows a program to include database manipulation functions. If this is the case, then SQL statements can be submitted directly from the CGI program, just as in a client/server application.

There are also programming language-specific interfaces which provide a consistent interface to a variety of DBMS's. For example, there are libraries available for accessing various DBMS's from a Perl script. For accessing a Sybase database from a Perl CGI script, there is a library called SybPerl. For Oracle, there is OraPerl, and for Informix, IsqlPerl.

Alternatively, Microsoft's Open Database Connectivity (ODBC) standard can be used to manipulate any ODBC-compliant database using a consistent interface.

5. Advantages of the CGI Approach

One advantage of using CGI to access a database is that it is a standard widely supported by virtually all HTTP servers. This makes CGI programs extremely portable. In addition, it is an highly flexible interface, in that almost any programming language supported by the operating system can be used to write a CGI program. Furthermore, since it is currently the most widely used method to access a database from the Web, there are more programmers knowledgeable about using CGI and more information available than some of the newer database access methods. There are also many public domain CGI programs available and a variety of development tools for developing CGI applications. CGI is also relatively easy to use.

6. Disadvantages of the CGI Approach

The most serious drawback of the CGI approach is that a new process is spawned for each incoming CGI request. This method is very inefficient, both in the amount of time it takes to spawn a new process, and in the overhead required by the server to manage many concurrent processes. At a busy Web site, the number of concurrent CGI

processes can quickly consume system resources, including server memory. Thus, the CGI approach is not scaleable.

In addition, communication between the HTTP server and the CGI process via STDIN and STDOUT is slow. If the CGI program is returning large amounts of data, this will incur a great deal of overhead.

The third disadvantage to using CGI is that a new connection to the database must be established for each request. It is more efficient to maintain an active connection to the database between requests.

Finally, there are security risks associated with CGI that can make a system susceptible to hackers. The details of these types of security risks are beyond the scope of this thesis.

C. THE SERVER API APPROACH

1. What is a Server API?

In response to the performance problems of the CGI approach some of the major HTTP server software vendors, such as Netscape and Microsoft, have devised an alternative approach to CGI. This alternative approach consists of a programming interface that is included with the HTTP server software which allows programs to be run as part of the HTTP server process, instead of as a separate process as with CGI. Like CGI, Server API is an example of a Web/database approach in which the program runs on the Web server. Netscape's implementation of this concept is called Netscape Server Application Program Interface (NSAPI), and Microsoft's is called Internet Server Application Program Interface (ISAPI). Both companies claim that the Server API approach offers superior performance to CGI.

In order for a program to run within the HTTP server's process space, it must be compiled into a dynamic link library (DLL) so that it can be dynamically linked at run time. When the program is invoked by a client request, it is loaded into memory in the

same process space as the HTTP server. Once the program has completed executing, it can remain in memory so that it can be quickly accessed when it is needed again. After it has not been accessed for a certain amount of time, the program will be unloaded from memory to free up resources. In this way, the most frequently accessed programs will remain in memory, while those programs which are rarely used will be unloaded.

Once it is loaded into memory, a Server API program can be executed by more than one client at a time. Therefore, it is necessary for the application to support multi-threading and perform garbage collection.

Because the application shares the same process space with the HTTP server, communication through STDIN and STDOUT is no longer necessary. The HTTP server communicates with the application through a shared buffer space. In addition, CGI environment variables are made available to the application through procedure calls. Figures 6 and 7 illustrate the differences in the way processes and communication are handled in CGI and in Server API.

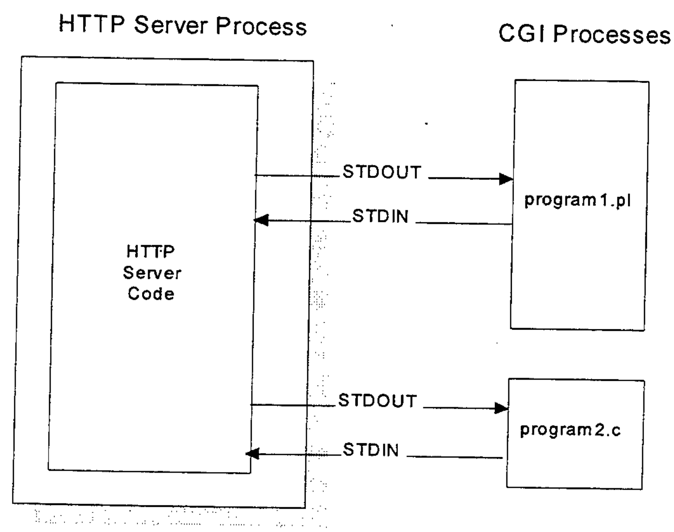


Figure 6. Process Configuration and Communication in CGI

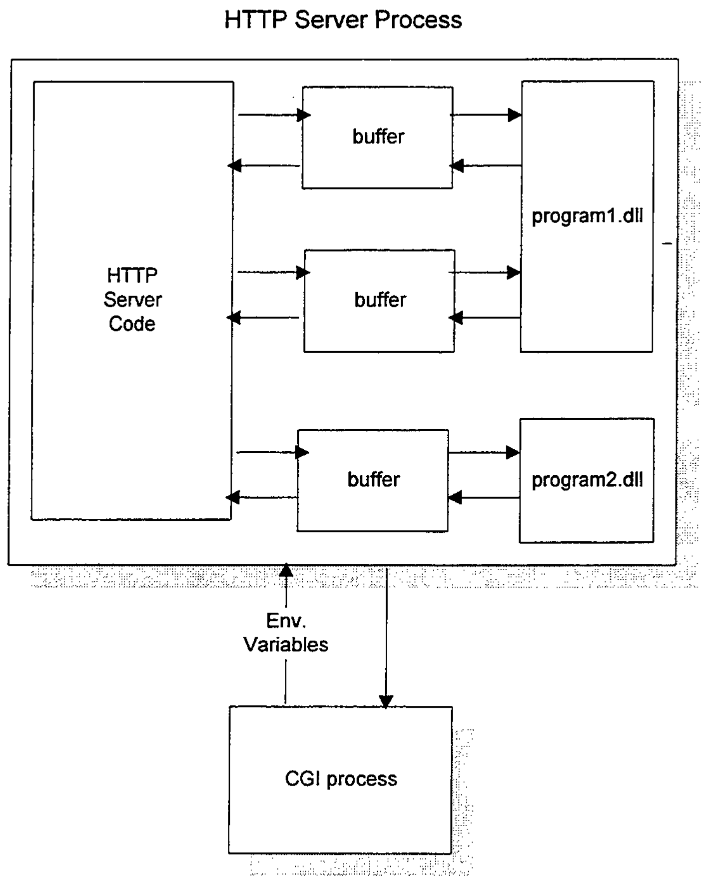


Figure 7. Process Configuration and Communication in Server API

A Server API application is invoked in much the same way as a CGI program is. Parameters are passed as name-value pairs, and the program is referred to with an extension of .dll (or .so in the case of the Netscape server). Data is returned to the client in the same way, with a MIME header to indicate what type of output it is. Database access is also achieved in the same manner as in CGI.

2. Advantages of Server API

The most significant advantage of the Server API approach over CGI is in performance and scalability. The overhead of spawning a new process for each incoming

request is eliminated and the Web server will not become burdened with multiple concurrent processes. In addition, the necessity for using STDIN and STDOUT streams for communication between the HTTP server and the application is also eliminated, thus speeding up the process. This is an especially important aspect in database applications, which may often return large quantities of data. Benchmark tests have indicated that the Server API approach performs far better than CGI in connections per second, response time, and throughput [Ref. 1].

An additional performance benefit is that a Server API application can request information from the client instead of passing all possible information. This eliminates passing information that is unnecessary and gives the programmer greater control.

Another advantage to this approach is that existing CGI applications can be converted to Server API applications relatively easily. The application must be made thread-safe, so that it can be executed by more than one client at a time. This can be accomplished by using critical regions and semaphores in modifying global variables. Minor modifications must also be made in the way input and output is handled, but essentially, the logic contained in existing CGI programs can be retained. This means that current applications do not need to be completely re-engineered in order to take advantage of the performance improvements of the Server API approach.

Finally, because the application is more tightly integrated with the HTTP server, it is able to access to internal server functions that are not available to CGI programs.

3. Disadvantages of Server API

One disadvantage to this approach is that Server API applications are specific to a particular HTTP server and operating system. The portability of CGI is not retained in this approach, and not all HTTP server vendors offer an API, so anyone wishing to take advantage of this technology must purchase their HTTP server from one of the vendors that does offer this option.

Another disadvantage is that because Server API programs share the same memory and process space as the HTTP server, a poorly written application could easily

overwrite the memory of another application, or that of the HTTP server itself, causing the entire server to crash. It is essential that proper care is taken to ensure that the code is thread safe and performs garbage collection.

D. ORACLE'S WEB REQUEST BROKER/PL/SQL SOLUTION

In response to the demand to access databases through the Web, several of the major DBMS vendors have begun to develop their own integrated solutions to building a Web/database interface. These vendors include Sybase, Informix, and Oracle. This section will focus on Oracle's solution, as it will be used in the case study presented in Chapter V.

Oracle Corporation has developed its own Web Server architecture which allows a user to specify a URL which will invoke a Procedural Language/Structured Query Language (PL/SQL) stored procedure in an Oracle database. The stored procedure can in turn perform database queries and return data in an HTML document to the user's browser. This is an example of a Web/database interface method in which the database access takes place on the database server itself.

1. Architecture

Oracle Web Server 2.0 consists of two components; an ordinary HTTP server, called the Oracle Web Listener, and a dispatch mechanism called the Web Request Broker (WRB). Incoming requests are received by the Web Listener and immediately handed off to the Web Request Broker for processing. If the request does not specify a WRB service then it is returned to the HTTP server to determine if it refers to a static HTML file or a CGI program. The WRB is configured to map virtual directories to specific server extensions. These server extensions are similar to the server API applications discussed in the previous section. They consist of a common Execution Engine (WRBX) and a shared DLL, which is specific to the particular application. The DLL programs are referred to as *cartridges*. Oracle provides an open API to the WRBX

engine so that customers and vendors can build custom server extensions. Web Server 2.0 comes with some programmable cartridges already built in. These include a Java Interpreter cartridge and a PL/SQL agent cartridge which allow for database access. An example of a system cartridge being developed by an outside vendor is Verifone's Virtual Point of Sale (VPOS) cartridge which will allow Web applications to perform electronic payment transactions. The architecture of Oracle Web Server 2.0 is depicted in Figure 8.

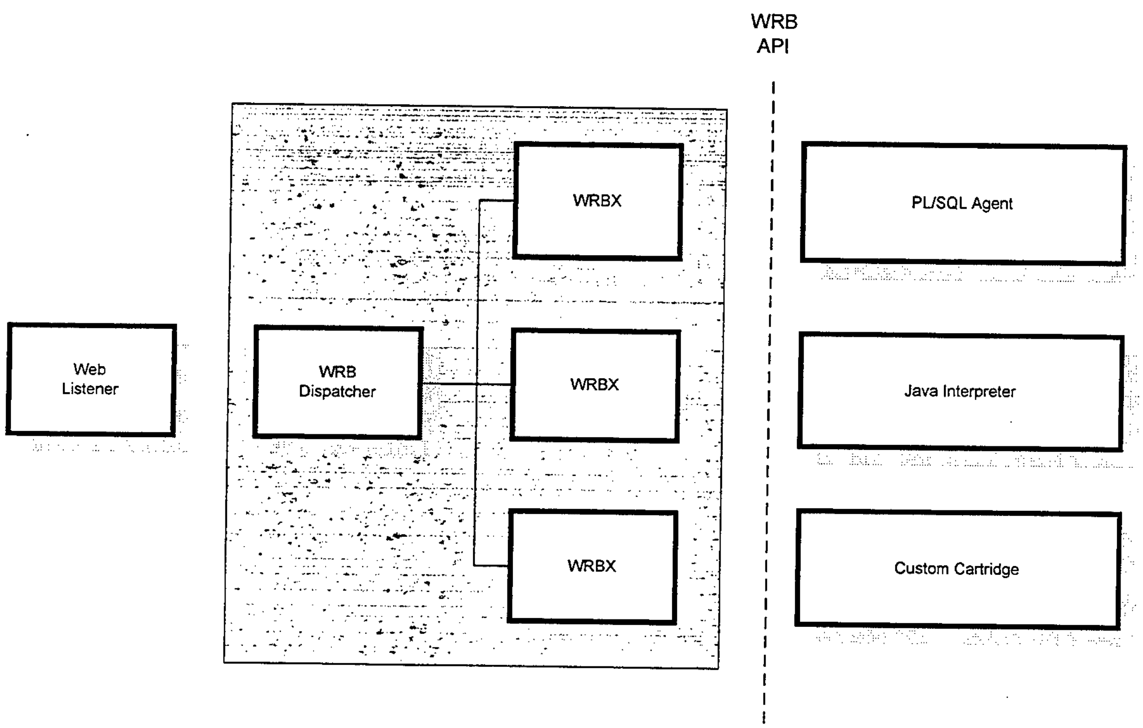


Figure 8. Oracle Web Server 2.0 Architecture

2. Database Access

The PL/SQL cartridge that is included with Web Server 2.0 allows a stored procedure in the database to be invoked. The Web Server administrator defines Database Connection Descriptors (DCD's) which map virtual directories to a specific database. Therefore, if an organization has multiple Oracle databases, different DCD's can be set

up for each database. In this case, separate instances of the PL/SQL agent will exist for each DCD. The keyword "owa" must appear in the URL to specify the PL/SQL agent, and it must be preceded by a reference to a particular DCD. What comes after the keyword owa is interpreted as a PL/SQL program. For example, in the following URL:

```
http://www.mycompany.com/mydcd/owa/myproc
```

"www.mycompany.com" specifies the Web server. The keyword "owa" indicates to the WRB that the PL/SQL agent services are being requested using the DCD "mydcd". The stored procedure "myproc" will then be executed in the database specified by "mydcd". The PL/SQL agent launches the stored procedures and provides a mechanism for the stored procedures to return output to the client's browser in the form of HTML documents.

Web Server 2.0 also comes with a PL/SQL Web Toolkit which is a collection of packages, procedures, functions and utilities which allow the PL/SQL programmer to produce HTML documents as output. Detailed examples on the usage of these tools will be presented in Chapter V.

3. Advantages of WRB/PL/SQL

There are substantial advantages to this approach. The use of stored procedures provides a native access to the database. For data-driven applications, it makes sense to store the programs as part of the database. The PL/SQL language was designed for database manipulation, so it has many powerful capabilities that are not available in other languages.

In addition, this method of accessing a database through the Web is much faster than CGI, although the server architecture also provides the capability to run CGI programs for backwards compatibility. The ability to create cartridges also provides a

mechanism similar to the server API method for creating custom server extensions, but without the danger of poorly written code crashing the HTTP server.

The PL/SQL agent remains connected to the database between requests, so a new connection does not have to be established for each request. This solution is also more robust and provides greater scalability than the other methods. It is more appropriate for industrial-strength applications.

Finally, by choosing this solution, you are not necessarily bound to Oracle's HTTP server. With Web Server 2.0, Oracle provides a WRB Adapter, which will allow the WRB Dispatcher to work with other HTTP servers. Currently, only the National Center for Supercomputing Applications (NCSA) and Apache servers are supported, but Oracle plans to provide support for many others in future releases.

4. Disadvantages of WRB/PL/SQL

The primary disadvantage to this approach it is specific to a particular DBMS. If an organization is not running an Oracle database, then this solution is not available. In addition, this approach can be more expensive than the other approaches.

E. CLIENT SIDE DATABASE ACCESS VIA JAVA'S JDBC

Another possible approach to building a Web/database interface is to have the database access occur directly from the client. This can be done in a variety of ways, including the use of browser-extensions. One of the more promising and interesting ways that client-side database access can occur is through the use of Java applets and the Java Database Connectivity (JDBC) protocol.

1. What is Java?

Java is a programming language developed by Sun Microsystems primarily for Internet programming. Sun's James Gosling [Ref. 2] describes the Java language as a

"simple, object oriented, and familiar...robust and secure...architecture-neutral and portable...high performance...interpreted, threaded and dynamic" language. Java is similar in syntax and capability to C++, but it is somewhat simplified. Its popularity for Web programming lies in its platform independence. Java is not compiled like C++; it is interpreted. A Java program is converted into *bytecode*, which can then be run on any machine under any operating system that has a bytecode verifier. The bytecode verifier simply interprets the bytecode and executes the program. Bytecode verifiers are available and can be downloaded for most operating systems, and in the future, will probably be included with many operating systems. This platform independence makes Java a very powerful language for Web programming.

The Java language can be used to write programs called *applets* which can be downloaded as part of a Web page and executed in the client's browser. These applets can then be used to access a database directly from the client by using JDBC.

2. What is JDBC?

As Java becomes more and more popular as a Web development language, developers have begun to look for ways to access a database from a Java applet. In response, Sun Microsystems has developed an API called JDBC which will provide a consistent interface for accessing SQL databases from Java programs.

The JDBC interface is based on Microsoft's ODBC, which is a popular standard for database access in client/server applications. For consistency and security reasons, the JDBC API is written in Java, whereas the ODBC API is written in C.

The JDBC interface will allow programmers to define database connections, execute SQL statements, and process the results set. The results set can then be displayed in the client's browser.

In order to access a particular DBMS, however, a DBMS-specific driver is required. These drivers are expected to be provided by the DBMS vendors themselves, since they would like their databases to be accessible from Java applications. The JDBC specification was released by Sun Microsystems in June of 1996, and currently, there are

no DBMS-specific drivers yet available. They are still under development, but are expected to be on the market soon. There are JDBC-ODBC bridge drivers currently available, which simply convert JDBC calls to ODBC calls, and use the ODBC drivers to access the database. This is really only a temporary mechanism which will only be necessary until the JDBC drivers are available. Once these drivers are made available, Java and JDBC may become an extremely popular method of accessing a database through the Web.

In this scenario, the Web is really just a means for delivering an applet to the client's desktop, and the browser provides a framework in which to run the applet. The database access actually takes place directly between the client and the database. This is much more like a traditional client/server application. Figure 9 shows how a database can be accessed from a Java applet.

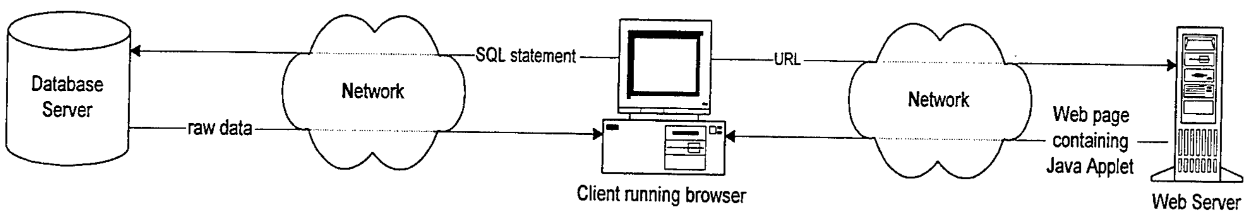


Figure 9. A Web/Database Interface Using Java and JDBC

3. Advantages of Java/JDBC

One advantage of this approach is that the database interface provided by JDBC is independent of the particular DBMS and connectivity mechanism. This means that the same code can be used to access a database whether it is an Oracle database or Sybase, or any other DBMS vendor which supports JDBC.

Java's platform independence is another benefit. The same Applet can be downloaded and run on almost any client. This is an extremely powerful feature.

In addition, Java has much more sophisticated capabilities for creating user interfaces. It is possible to display dialog boxes and a variety of other features that are not possible in any other way. This extends the capabilities of the Web/database interface to perform much like a client/server application, but without the difficulties of deploying and maintaining a client/server application.

4. Disadvantages of Java/JDBC

Because JDBC is a very immature technology, its performance is yet unknown. However, it is known that Java is much slower than compiled languages, because it must be interpreted on the client machine.

Currently, the only way to implement a Java/JDBC interface is by using a JDBC-ODBC bridge, since the DBMS-specific drivers are not yet available. This mechanism will be slow, as it involves multiple layers of software.

In addition, many people feel that Java has inherent security risks. Although the developers of the Java language have taken great pains to insure that the language is secure, there is always a danger in downloading a program onto a client machine and allowing it to run.

IV. RELATED ISSUES

A. OVERVIEW

Along with the vast potential of the Web/database technology comes many new challenges. The very features that make the Web a flexible and effective platform for accessing data also pose new problems that have yet to be overcome. Some difficulties exist in the areas of speed and performance, concurrency, and security. This chapter discusses the issues relating to each of these areas.

B. SPEED AND PERFORMANCE ISSUES

Speed and performance is a critical area in Web/database applications, because the success of any application depends heavily on how user-friendly it is. Even if an application is easy to use in other respects, slow response times will discourage users. This is particularly true in a Web environment, where an impatient user can easily abandon the application by clicking the Stop button in the browser or by simply switching to another Web page. Many of the same rules apply to Web/database applications as in ordinary database applications, but there are also additional factors to consider.

The first factor is that in a Web/database interface, there are multiple servers, network connections, and programs involved. Therefore, performance problems can occur at various levels, and can be difficult to isolate. Because of this, it is important to have well-defined interfaces and interactions between the various software components, and applications should be designed and tested carefully with performance in mind.

The connectionless nature of Web/database interfaces also causes challenges in the area of performance. In a normal client/server application, the user remains connected to the database throughout the session. In a Web/database application, however, each Web page is independent, and a new database connection must be established for each page. For example, if a Web page allows a user to retrieve some

data, view it, and then update it, the application must first connect to the database in order to retrieve the data, and then re-connect to update it. There is a great deal of overhead involved in establishing and terminating database connections. In addition, many DBMS's perform optimization on a per-session basis. Continually connecting and disconnecting from the database invalidates this type of optimization. Some progress has been made toward a solution to this problem, however. Oracle's Web Server software circumvents the need to repeatedly connect to the database by using a daemon (the PL/SQL Agent) which remains connected to the database and simply passes requests through.

The third reason why performance is a problem in Web/database applications is that the type of connection, platform, and even physical proximity of the end users must be taken into account. While some users may have a high performance Internet connection and a powerful desktop computer, others may be dialing in by a modem, with a slow computer, or may be located halfway around the world. The application must be able to perform adequately for all of its intended users. Therefore, it is important to be aware of who the target users are, and to test a Web/database application on a variety of platforms, using different types of connections.

Factors that affect the speed of a Web/database application include the size of the tables being accessed and the amount of data being returned. As in traditional database applications, it can be extremely time consuming to search a large table. Indices can be used to improve response time. It is a good idea to partition the Web application such that large amounts of data will not need to be returned. Transporting large amounts of data over the Internet can be a bottleneck.

A third factor influencing the performance of a Web application is the importance of tuning the SQL queries and using them judiciously. Dwight and Erwin set forth three rules for optimizing performance in a Web/database interface:

- Do as much work as possible per connection to the database server.
- Do as much work as possible per SQL statement.
- Filter the results inside the database server as much as possible.

The first point indicates the importance of performing as many queries or updates as necessary during each connection to the database server. This will minimize the number of connections that must be made during a session. The second implies that each SQL statement should be written to do as much data manipulation as possible. If one large SQL query can produce the same results as two smaller queries, then the larger query should be used. The third point says that it is a good idea to use SQL's extensive data manipulation capabilities to limit the amount of data returned, rather than trying to manipulate the data in the program. Dwight and Erwin also suggest formulating the SQL statements to be used and testing them using the command line interface before trying to incorporate them into the program. [Ref. 3]

The fourth factor is the speed of the database server and the Web server. Both Web servers and database servers must be able to handle large amounts of I/O (input/output). Therefore, it is important that they have fast disk drives as well as adequate memory.

The fifth and final factor that has an impact on the performance of a Web/database application is the use of graphics. While many popular Web sites make use of graphics to enhance the user interface, graphics can incur a great deal of overhead. If a Web application is intended for a wide variety of users, it is important to use them sparingly. It may even be necessary to maintain a graphics-free version of a Web application in order to accommodate all users. Although most browsers can be configured to display text only, Web applications which rely heavily on graphics will be difficult to use without the graphics.

C. CONCURRENCY ISSUES

In any Web/database interface that allows the users to update the database, concurrency will be a concern. The absence of a persistent database connection in the Web environment causes serious concurrency problems. In a client/server application, the DBMS handles concurrency through the use of atomic transactions and locking techniques. However, because the user does not remain connected to the database in a

Web/database application, these mechanisms can not be relied upon. The HTTP protocol was simply not designed for transaction processing. Thus, it is up to the programmer to ensure that database concurrency is maintained.

For example, consider a Web interface to a warehouse inventory database which allows warehouse employees to check and update the number of a given item in stock. Suppose two warehouse employees receive orders for a particular computer model. Employee A's request is for 15 computers, and Employee B's request is for 10 computers. Employee A checks the inventory through the Web interface and finds that there are 20 in stock. While Employee A is editing the number of computers in stock to reflect that there will only be five computers left, Employee B also checks the inventory and finds that there are 20 in stock. When Employee A submits the update, Employee B's information becomes inconsistent. Now when Employee B tries to submit an update, Employee A's changes will be lost. The database will reflect that there are 10 computers left, when there really are not even enough to satisfy both requests. If both employees had been connected to the database throughout the transactions, this type of concurrency conflict would have been prevented, but since they must establish a new database connection for each interaction, the DBMS's concurrency control mechanisms can not be relied upon to prevent such problems. The Web/database application will need to perform updates in such a way as to keep multiple users from editing the same information at the same time.

There are various techniques that can be used to handle concurrency in a Web/database application. The type and complexity of the transactions to be performed must be evaluated, as well as the likelihood that a concurrency conflict will arise and the consequences of such a conflict.

Some applications only allow users to perform database inserts. Examples of such applications are a guest comments log, or an order form. In this type of application, the use of a unique key should be sufficient to ensure that concurrency conflicts will not occur. A unique key can be generated by using user information, time and date information, a sequence number, or some combination thereof. This should prevent users from overwriting each other's records.

For applications with more complex transactions than a simple insert, more sophisticated concurrency mechanisms must be employed. One solution is to verify each value before updating. This means that once a user retrieves some data, edits it, and submits and update, the program must re-check the original values of each of the fields to be updated against their current values in the database. If any of the values do not match, this implies that the data has been updated, and an error message must be returned to the user. This method works well for editing a small number of fields.

A variation on this method is the use of timestamps. Timestamps are more applicable when a larger number of fields must be updated, and it would not be easy to verify the values of each field. This method requires that the database includes a timestamp field. Each time a record is updated, the associated timestamp must also be updated to the current time. Then, when a user retrieves a record, the timestamp is also retrieved. When an update is submitted, the timestamp that was retrieved is compared against the current timestamp. If they are different, then the record has been updated since the data was originally retrieved, and again, an error message must be displayed. This method has the same effect as the double-checking method, but the checking is isolated to one field instead of many.

One other way to prevent concurrency problems in a Web/database application is by including a checkout field in the database. This would be a Boolean type of field with values true or false. When a user attempts to retrieve a record, the checkout field is first checked to see if it is false. If so, it is set to true and the record is returned to the user for editing. Then, when another user attempts to download the same data, the checkout field will indicate that the record is already in use and an error message can be displayed before the user has already made changes. This prevents the scenario in which a user makes changes to some data, only to find out that the data has since been updated. The user is notified ahead of time that the data is in use. However, it does pose other problems. For example a user could download some data, and then go to lunch, leaving the record locked for a long period of time.

All of these solutions are less than ideal. While they may be adequate for simple applications, they are not sufficient to address the requirements of complex transaction

processing systems. DBMS manufacturers have gone to great efforts to provide sophisticated concurrency control mechanisms with their software, but Web applications are unable to take advantage of these capabilities. It is expected that these problems will eventually be solved with future versions of the HTTP protocol and other software, but for now, it is up to the Web developer to compensate for the absence of a persistent connection.

D. SECURITY ISSUES

Security has become an extremely complex and critical issue in light of the rapid growth in Internet technology. The Internet was designed to be an open, flexible system; it does not provide any privacy or security in itself. However, with the expansion of the World Wide Web and its increasing use as a business tool, Internet security has become a necessity. When an organization connects its internal Local Area Network (LAN) to the Internet, it opens itself up to a variety of risks such as viruses, hackers, and eavesdroppers who intercept Internet transmissions. By connecting its corporate database to the Web, an organization risks exposing critical information. The challenge is to tightly control what information is made available and specify the desired users, as well as preventing individuals from intercepting valuable information as it is transmitted over the Internet.

Data stored in a database is inherently more secure than data stored in flatfiles. While static HTML files on a Web server rely on the security of the operating system and the HTTP server, information in a database is protected by the security mechanisms of the DBMS. There are risks, however. The most serious risk is that a hacker may somehow gain access to the database passwords and break into the database to alter, destroy, or download unauthorized data. Another risk is that an unauthorized user may gain access to database through the Web/database interface. It is necessary to control what users are allowed to access the Web interface. Finally, because of the insecure nature of Internet data transmissions, it is possible for an individual to intercept data from the database as it is transmitted from the Web server to the client or userid and password information as it is transmitted from the client to the Web server.

A combination of security mechanisms are used to protect data from these types of intrusions. Of course, the first line of defense is the built-in security features of the database. It is important that the userid(s) that clients will be connecting to the database as are low-privilege accounts, with only the capability to read and write the necessary database objects. For a Web/database interface that is read-only, the userid should be limited to read privileges on only the tables that are referenced in the application. This is accomplished by using the SQL "GRANT" statement. For update privileges, it is possible to specify an even finer granularity, such as a particular column. Some DBMS's allow privileges to be granted to groups or classes of users, as in Oracle's User Roles mechanism.

In addition to the DBMS's native security features, there are other methods of protecting a database. These include authentication, encryption, and the use of firewalls.

1. Authentication

Authentication refers to the process of ensuring that both the user and the server are who they claim to be. This prevents unauthorized users from gaining access to the database through the Web/database interface, and additionally prevents Internet Protocol (IP) address *spoofing* in which an IP address is falsely mapped to a impostor server, so that information such as userids and passwords can be collected from individuals.

Authentication of the client is usually accomplished by way of a userid and password combination, but other mechanisms can be used, including hardware solutions such as Smart Cards, which uniquely identify an individual. Authentication can be used to specify which users have access to a particular Web page. It can also be used to specify different levels of access. For example, some users may only have read access to a Web/database interface, while others may have update capabilities. The level of complexity necessary depends upon how sensitive the data is, who the intended users are, and whether or not updates are allowed. Some applications do not require any authentication at all, but for many applications, authentication is critical.

Userids and password schemes can be implemented through the HTTP server, the Web server file system, or the database itself. Most HTTP servers come with an authentication scheme which allows the Web server administrator to define individual users as well as groups of users with various levels of access. The userid and password files must be stored in a protected directory. In this case, the client would have to supply the a correct userid and password to even access the Web page, and all users would then log into the database under the same userid. Another alternative is to use the DBMS's userid and password scheme and create an account in the database for each user. In this way, any user could access the Web page, but only those who could supply the correct password and userid could connect to the database. This method requires more maintenance, however. A new userid must be created for each user or group of users, and the appropriate privileges must be granted.

2. Encryption

Another technique used to provide security in a Web/database application is encryption. Encryption is used to make data and passwords unreadable as they traverse the Internet. This is important if the data in the database is sensitive or confidential. It is also essential that password and userid information are encrypted as they are transmitted from the client to the Web server. The information is encrypted prior to transmission using a numeric key such that it can only be decrypted by the intended receiver. There are two general types of encryption: symmetric and asymmetric.

With *symmetric encryption*, also known as *private key encryption*, the same numeric key is used to encrypt and to decrypt a message. In order for this to work, the sender and the receiver must agree on the secret key in advance. This turns out to be logistically difficult. To solve this problem, *asymmetric*, or *public key*, encryption was developed. In public key encryption, each host has its own secret key, as well as a public key, which is made available in a public directory. When one host wants to send an encrypted message to another host, the sender uses the receiver's public key to encrypt the message. Because a user's public key and private key are related mathematically, the

receiver is able to decrypt the message using his/her private key. In this way, a message can be encrypted such that it can only be read by the intended user.

Standards for secure data transmission are beginning to emerge as Internet software vendors such as Netscape, Microsoft, and Oracle are now including encryption capabilities with their products. Netscape's Secure Sockets Layer (SSL) encrypts HTML documents using secure HTTP, as well as providing authentication and ensuring data integrity. The federal government, however, has mandated the use of Fortezza, a more robust encryption and authentication mechanism.

3. Firewalls

Firewalls are used to keep outsiders out of an organization's internal network. The term firewall does not refer to a particular technology or capability, but rather to software that employs a set of general principles and techniques to prevent intrusions from unauthorized users. The firewall forms a perimeter around an organization's network by creating a central point of entry to the network and by restricting the users that are allowed access and the types of services they are allowed to use.

Generally, an organization's public Web server sits outside of the firewall and accepts incoming requests. These requests must then be passed through the firewall to the internal network, where the database resides. This configuration is illustrated in Figure 10.

There are two basic categories of firewalls. The first is referred to as a packet filter. A packet filter is a type of router that filters out traffic from all hosts except those that are known to be trusted. This filtering is based upon the IP addresses of the sender. By using this method, and organization can ensure that only trusted hosts will have access to its internal network.

The second basic type of firewall is known as a proxy server or application gateway. Proxy server firewalls generally use a dual-homed host in which the host machine is configured to have two network interfaces; one to the internal network and one to the Internet. This creates a barrier between the inside and the outside. The proxy

server is configured to only allow certain services. Incoming packets are evaluated individually and either discarded or allowed to pass through. For legitimate requests, the proxy acts as an intermediary between the Internet interface and the internal network interface. Firewall software also traces all incoming and outgoing requests and keeps log files of these requests. These log files should be analyzed on a regular basis for unusual or suspicious patterns.

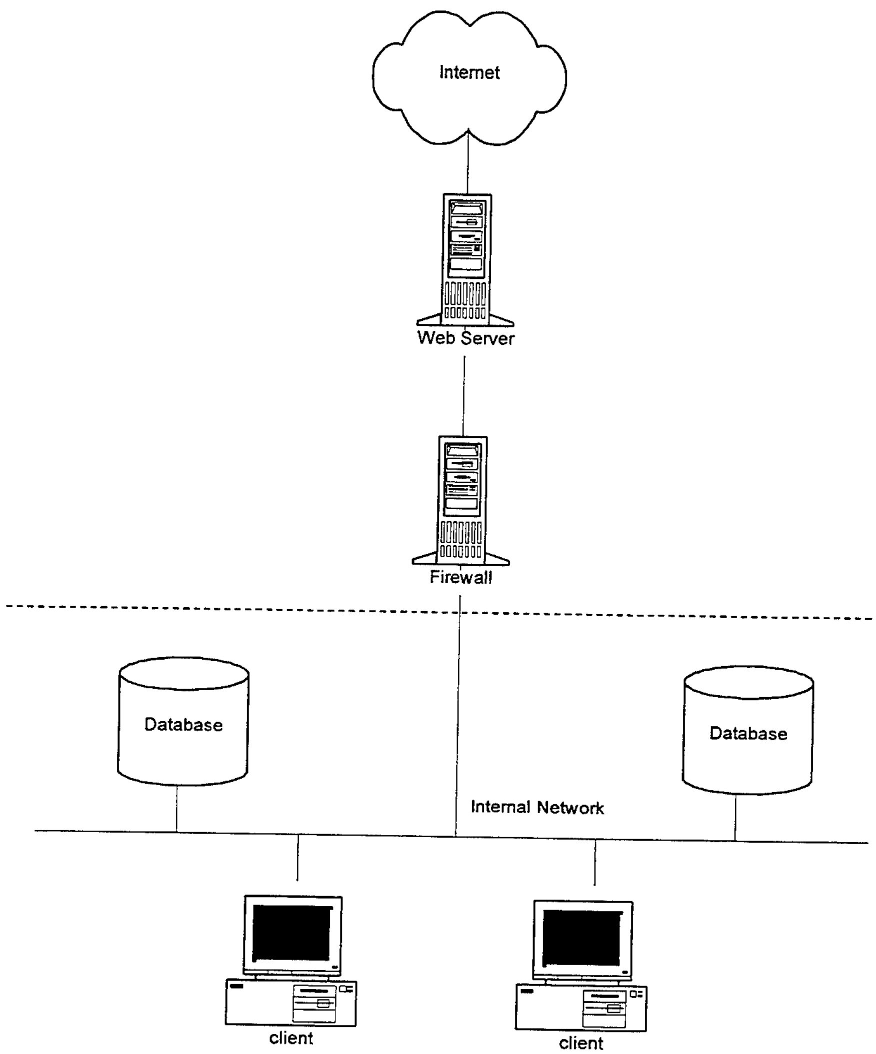


Figure 10. A Network Firewall Configuration

V. CASE STUDY: OHA WEB/DATABASE INTERFACE

A. BACKGROUND

The Overseas Housing Allowance (OHA) is an allowance paid to military members who are stationed overseas in order to supplement their housing costs. The particular country, location within a country, rank, and dependency status (with or without dependents) determines the amount a member is eligible for. The Per Diem, Travel and Transportation Allowance Committee (PDTATAC) is responsible for setting these rates.

The Defense Manpower Data Center (DMDC) currently maintains a legacy OHA database system on the Naval Postgraduate School's International Business Machines (IBM) mainframe in support of this program. The purpose of this system is to make OHA rates and related information available to Defense Finance and Accounting Service (DFAS) centers so that military members eligible for the allowance can be paid, as well as to produce quarterly reports for PDTATAC which provide information used to set new rates, and maintain historical archives of this data. PDTATAC maintains their own database and sends bi-weekly updates to DMDC's database in the form of a transaction file.

In order to eliminate the redundancy of maintaining two separate databases and to modernize the system and add functionality, DMDC has been tasked with the development of a new OHA database system which would combine the capabilities of DMDC's and PDTATAC's existing databases. The proposed system will consist of an Oracle 7 database with a user interface written in Visual Basic 4.0. It will have built in reporting and publishing capabilities developed using Visual Basic's report tool, as well as an artificial intelligence component written in AionDS that will be used to apply heuristic rules to the process of setting rates. This new system is scheduled to begin beta testing late in 1996. The current system will be run in parallel for several quarters to

ensure that the new system is working smoothly. The Entity-Relationship diagram and schema for this database are illustrated in Appendices A and B.

In addition to the existing database, PDTATAC currently maintains a Web site that makes OHA rates and various other information available to military members. This Web site resides on the Defense Technical Information Center (DTIC) Web server. In order to make OHA information available on this Web site, PDTATAC downloads the current rates from DMDC's database bi-weekly, reformats it, and transfers it to the DTIC server, where it is accessed by a CGI Perl program.

Once the new database is operational, it would make sense to eliminate these steps and tie the Web page directly into the Oracle database. This way, the data would not have to be stored in two different places in two different formats, and maintenance would be much simpler. In addition, the most current information would always be available. This is important, because the rates change frequently due to currency fluctuation.

This case study will consist of building and documenting a prototype of the Web/database interface to the new OHA database.

B. ANALYSIS

There are several important aspects of this application to consider before deciding on an approach and designing the interface.

1. Integration With Existing Web Site

One important factor is that the Web page that is developed must eventually be integrated into PDTATAC's existing Web site. Viewing OHA data is one of several menu options available on PDTATAC's Web site, and while the newly developed Web/database interface will replace this menu option, the rest of the existing Web page will remain intact. Therefore it is important when designing the new OHA Web page to maintain a look and feel that is consistent with the existing Web site. This means using

the same background and graphics, the same style for titles, and the same type of form controls.

2. Users

Another important factor to consider is who the users of the Web page will be. The primary users of this Web page will be military members who are or will be stationed overseas. Their level of sophistication will most likely vary from experienced computer users to the extremely inexperienced.

3. Speed and Performance

The size of the tables to be accessed and the quantity of data returned must also be considered. The tables in this case are relatively small. The largest table is under 1000 rows. These tables will change in size as locations are added and deleted, but they are not likely to change drastically in size. Therefore, the queries should not take a substantially long time to run. In addition, the amount of data to be returned is not especially large. This is important because it can take a long time to return a large amount of data over the Internet.

4. Concurrency

This Web site will be view-only. There will be no updates through the Web interface. The Oracle database will be updated through the client/server interface described in Section A. This means that concurrency is not a problem for this Web site. Multiple users can view the information without causing a conflict.

5. Security

The information contained in the Web page is public information. Security measures such as authentication and encryption are not necessary for this system. A firewall will, however, be necessary to protect the database itself.

6. Database

Finally, the Oracle database has already been designed and is already in place. No new database will have to be created for the Web interface. Although the database was designed with a client/server application in mind, the tables can be accessed through the Web without any modifications and without the necessity of duplicating any information.

C. APPROACH

Because the database is in Oracle, Oracle's WRB/PL/SQL solution will be used for this Web/database interface. The following is a list of tools that will be used to develop the interface:

- Oracle Web Server 2.0
- Oracle PL/SQL Web Toolkit
- Oracle Developer 2000 Procedure Builder
- Oracle SQL Plus for Windows
- Oracle SQL Loader Utility
- Netscape Navigator 2.0
- Novell LAN Workplace Host Presenter (Telnet Utility)
- Novell LAN Workplace Rapid Filer (FTP Utility)

D. DESIGN

The first step in designing the interface is to examine the existing OHA Web page and determine what aspects should be retained and what areas can be improved upon. The screen views of the existing OHA web page are shown in Appendix C. While the new Web page should conform aesthetically to the standards of the existing page, there is some flexibility in the design of the interface.

One area which can be improved is in the way the information is presented. In the existing system, the user selects a location, rank, and dependency status, and is then presented with the OHA rates based upon these selections. It would be useful to display all of the rates for a particular location in a table format, by rank and dependency status. In this way, a user could compare the rates for various ranks and for both dependency statuses, without having to perform the query repeatedly.

Another improvement can be made in the way the user selects a location to view. In the current implementation, the user must first select a location by name, look up a location code, and then re-enter the location code so that the query can be executed. This is a cumbersome interface, and is unnecessary in the new database. The location code should be transparent to the user.

One area that will be retained from the existing system is the use of select list boxes to display menu selections. This is an effective way to present a list of locations for the user to choose from. It is easy to use and understand.

Based upon these observations, the new OHA Web page will consist of three screens. On the first screen, the user will select a country by name. On the second, the user will select a location within the selected country, again by name, and on the final screen, the OHA rates for that location will be displayed in a table format by rank and dependency status, along with other associated information including the effective date for the rates, Moving-In Housing Allowance (MIHA) and climate information. These screens are shown in Appendix D.

E. IMPLEMENTATION

Several steps are involved in the implementation of the OHA Web/database interface. These steps are outlined in Figure 11. The first step is to re-create the necessary tables in a test instance on the Web server. The new OHA database is being developed in a development instance on another database server, but because a firewall is not yet in place, the Web server does not yet have connectivity to the other database servers on DMDC's LAN. Therefore, the prototype will be developed with the HTTP server and the database on the same server machine. The tables were re-created and populated by running the SQL create statements and SQL Loader statements in the test instance.

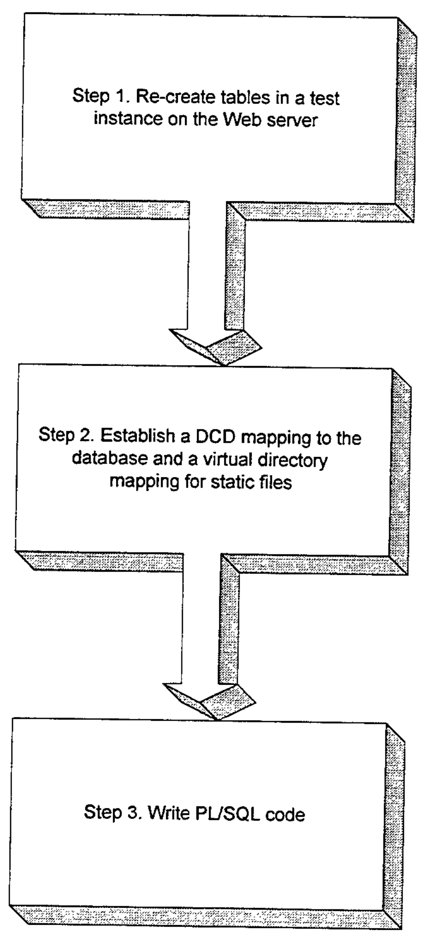


Figure 11. The Implementation Process for the OHA Web/Database Interface

The second step is to establish a DCD mapping to the test database. This will enable the WRB dispatcher to direct incoming URL's to the stored procedures in the database. The information needed to establish a DCD is a userid, password, and database instance. In addition to the DCD, a virtual directory mapping must also be established for static files on the Web server. This is necessary for the background GIF and other graphic files which will not be stored in the database. The Web server administrator establishes these mappings. The URL's are as follows:

```
http://206.39.184.1/oha/           for static files
http://206.39.184.1/oha/owa/       for PL/SQL procedures
```

The GIF files for the graphics files, which were copied from the existing OHA Web page, are then transferred to the corresponding directory on the Web server using the Rapid Filer File Transfer Protocol (FTP) utility.

Once the database is established and the graphics files are in place, the third step is to begin writing the PL/SQL code. The programs are written using Oracle's Procedure Builder, which is an editor that allows you to write and compile PL/SQL stored procedures in the database.

PL/SQL is very similar in syntax and structure to Ada. The program modules consist of procedures and packages. For this application, one package will be needed. This package will contain three procedures; one for each of the screens. Appendix E contains a listing of this PL/SQL package.

The first procedure generates HTML to display a select list of country names and prompt the user to make a selection. HTML is produced by using the hypertext procedures (HTP) and hypertext functions (HTF) that come with the Oracle PL/SQL Web Toolkit. These procedures and functions correspond one-to-one with HTML tags. They are essentially a programmatic interface to HTML coding. They take arguments and generate HTML tags. For example, the procedure call

```
htp.center('This text is centered');
```

will generate the following HTML:

```
<CENTER>This text is centered</CENTER>
```

This HTML will be sent directly to the client browser. This functionality is further illustrated in Figure 12.

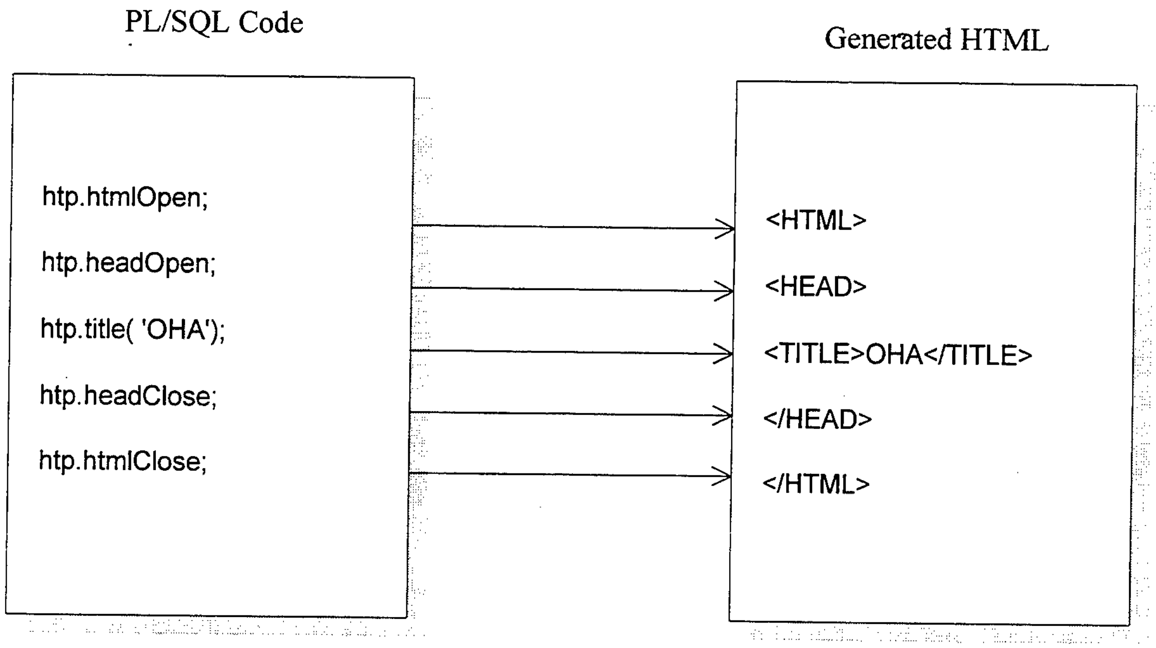


Figure 12. How HTML is Generated by PL/SQL Code

For every HTP procedure, there is also a corresponding HTF function. The difference between the hypertext procedures and hypertext functions is that the procedures send the output directly to the client browser, while the functions return the output to the calling program for further manipulation.

The corresponding HTF function call

```
ctext := htf.center('This text is centered');
```

generates the same HTML tags as the HTP version, but returns it to the variable ctext. This is useful for nesting HTML tags within other HTML tags. This method is used in the country selection procedure to create an image as an anchor as follows:

```
htp.anchor('index.html',
htf.img(curl=>url ||'/oha/earthico.gif'));
```

The HTP and HTF packages are used extensively throughout the country selection procedure to generate HTML header tags, body tags, and form tags. The HTML generated by the PL/SQL package is listed in Appendix F.

Database access is achieved by the use of a cursor. A cursor is a pointer to an area in memory containing rows from the database. The declaration for the cursor specifies the SQL select statement that is to be executed. In this case, a join between the country name table and the OHA rates table is used to select only those countries with valid OHA rates. The country name, country code, currency name, and currency code are the columns selected.

A cursor in PL/SQL is generally loaded by using a "fetch into" statement in which a variable or structure is specified as the target for the data. The variable or structure must be declared. One simple way to do this is to use the %rowtype attribute. If a cursor is declared called cntry_cursor, the target record structure can be declared as:

```
cntry_rec      cntry_cursor%rowtype;
```

the resulting structure will correspond to the columns in the select statement. In the country selection procedure, however, an implicit fetch and implicit structure declaration are used. The cursor is referenced as the control for a loop, as follows:

```
for cntry_rec in cntry_cursor
loop
...
end loop;
```

In this case, the cursor does not need to be explicitly loaded, and the structure cntry_rec does not need to be explicitly declared.

The select list on the country selection screen is created by using an HTML select list form control. HTP form procedures are used to produce the necessary HTML. The procedure call

```
htp.FormOpen(curl => 'OHA_PKG1.TSTOHA2',
cmethod => 'POST');
```

generates the HTML form tag which specifies what program to execute when the form is submitted. Another form procedure call,

```
htp.FormSelectOpen(CC,null,4);
```

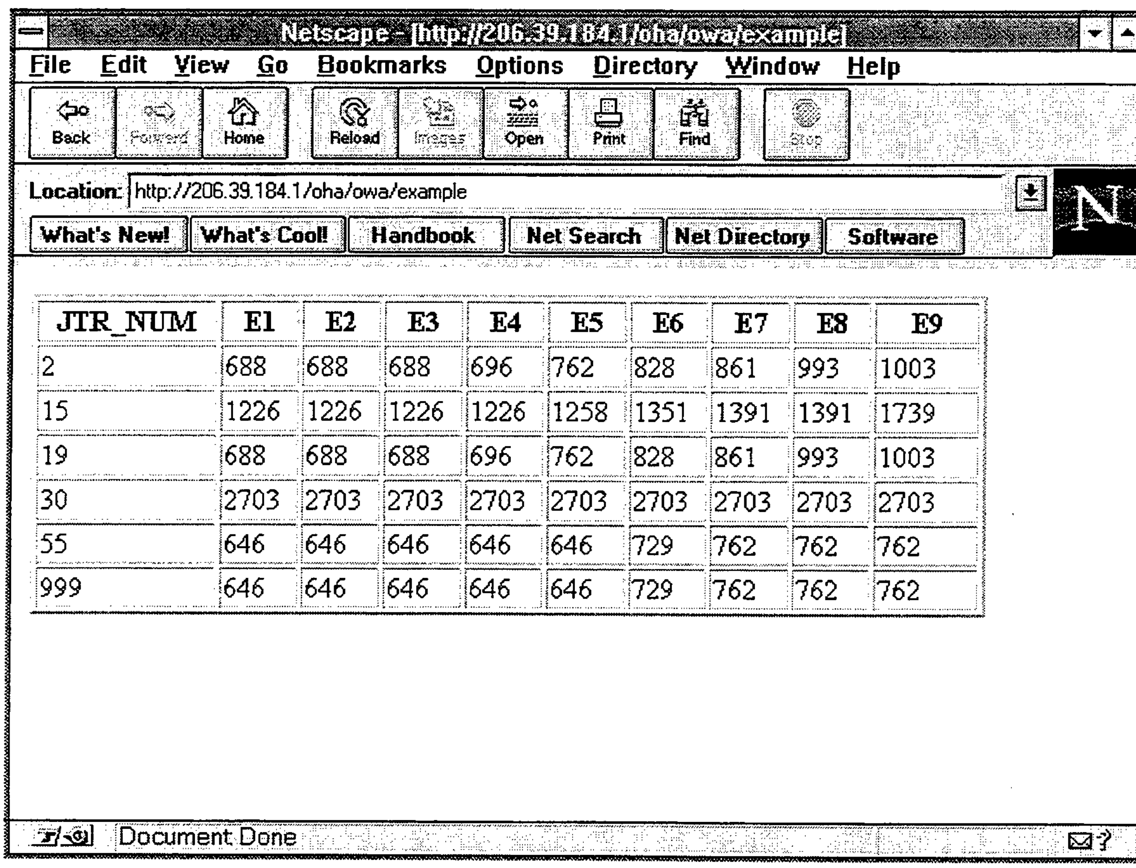
specifies the Select list control, and the variable to be passed to the next procedure. The select list is populated with country names from the database by using the htp.formSelectOption procedure call within the cursor loop. Elements of the cntry_rec structure are used to indicate the text to appear in the select list, as well as the values associated with them. It is the value that is submitted as a parameter to the next procedure when the user clicks on the Submit button. In this case, the data from several columns are concatenated together so that it can be passed to the next procedure as one string. This is necessary as only one value can be associated with each menu item.

The second procedure takes this parameter and parses it back into the individual values of country code, currency code, and currency name. This is accomplished by using the PL/SQL substr() function. This second procedure displays a list of locations within the selected country. It does this in a similar manner as the first procedure, using a cursor to select location name and other information from the JFTR location table and the OHA rates table. Again, a select list is used to display the menu. One interesting difference in this procedure is that it uses hidden fields to pass on information that was collected in the first procedure such as the currency code and the currency name. This is done by using the htp.formHidden procedure. When the user clicks on the submit button, the values in the hidden fields are passed to the next procedure along with any other values specified in form controls. By passing this information on in hidden fields, it saves the third procedure from having to do an additional select. This is an example of saving state information on the client side.

The third screen takes three input parameters; one from the select list, and two from the hidden fields. This procedure performs several select statements based upon data that was entered in the first two screens. The cursors in this procedure are explicitly loaded using "fetch into" statements, and the record structures are explicitly declared using the %rowtype attribute. The data that is retrieved is displayed in a table format using HTP table procedure calls such as htp.tableRowOpen, htp.tableHeader, and htp.tableData. These procedure calls are used to specify table column headings, row titles, and populate the cells of the table with data.

In addition to the HTP and HTF packages, the Oracle PL/SQL Web Toolkit also provides various utilities for performing useful functions. These utilities were very useful in the development of this interface. The owa_util.tablePrint utility allows for rapid prototyping by automatically generating an HTML table based on the database table name, column names, and a where clause. With one simple statement, a basic HTML table is generated. This utility does not provide much flexibility in the layout of the HTML table, but it does provide an easy mechanism for displaying database information on a Web page with very little effort. An example the use of this utility is shown in Figure 13. The table in this example is generated by the following statement:

```
rc := owa_util.tablePrint('KEY_JTR_OHA_RATE',
        'BORDER=2 WIDTH=90%',
        OWA_UTIL.HTML_TABLE,
        'JTR_NUM,E1,E2,E3,E4,E5,E6,E7,E8,E9',
        'WHERE CC = ''' || cc || '''');
```



JTR_NUM	E1	E2	E3	E4	E5	E6	E7	E8	E9
2	688	688	688	696	762	828	861	993	1003
15	1226	1226	1226	1226	1258	1351	1391	1391	1739
19	688	688	688	696	762	828	861	993	1003
30	2703	2703	2703	2703	2703	2703	2703	2703	2703
55	646	646	646	646	646	729	762	762	762
999	646	646	646	646	646	729	762	762	762

Figure 13. Example of Using the owa_util.tablePrint Utility

Another useful utility is the owa_util.signature procedure. This utility creates a signature line at the bottom of the HTML document that provides a hypertext link to the source code that generated the screen. This is a useful feature during the development phase, particularly if multiple developers are working on a Web application.

Other utilities include pattern matching facilities, the ability to retrieve CGI environment variables, image map handling capabilities, and mechanisms for manipulating HTTP Cookies.

F. INTEGRATING THE PROTOTYPE WITH THE PRODUCTION SYSTEM

Once the OHA database is operational, it will be relatively simple to integrate the prototype Web interface with the production system. The database will ultimately reside in a transaction processing instance. By this time, the firewall will be in place, with the Web server outside of the firewall, and the database servers inside the firewall. The PL/SQL code must be copied to the new database instance and recompiled. A userid will be created specifically for Web access, and it will be granted limited privileges, so that only the necessary tables and views can be accessed. Then, the DCD on the Web server must be modified to point to the new database instance and the new userid. There may be some minor changes to the PL/SQL code, such as referencing a view instead of a table, but essentially, the application is ready to go online.

G. ANTICIPATED FUTURE ENHANCEMENTS/MODIFICATIONS

There are already some areas in which changes or additions are anticipated for this Web/database interface. One area that will require modifications is in the transition to local currency. Currently, the OHA rates are prescribed in US dollars. This means that every time the exchange rate changes for a particular currency, the rates must change for all of the locations that use that currency. In order to separate changes based on currency fluctuation and changes based on actual changing housing costs, the rates will be prescribed in local currency. The OHA rates table will then have two views; one in local

currency and one in US dollars. The implications for the Web interface are that the user should have a choice of whether to view the rates in the local currency for a location, or in US dollars. In cases where the local currency is US dollars, this will not apply. This choice can be implemented by placing an icon or button on the final screen which will allow the user to toggle between local currency display and US dollars. When the button is clicked, the procedure will be re-executed and the Web page will be re-loaded with the opposite data. State information will have to be embedded in the button to indicate which state the user is currently in.

Another area in which the interface can be expanded upon is by displaying historical data. In many locations the rates change frequently, and it would be desirable to allow the user to view the changes in these rates over a period of time. The OHA rates table will contain historical information, as indicated by effective date. A view will be used to indicate the current rates for each location. By drawing information from the historical table, a Web page could display the changes in rates for a location over a period of time. This would have to be another menu item on the Web page, and it would probably require the user to specify a rank and dependency status, as it would be too much information to try to display historical rates data for all combinations of rank and dependency status on one Web page.

One final area of expansion for this Web interface is in the download capability for DFAS centers. Currently, the DFAS centers download updated information twice a month from the mainframe OHA system. Once the new database is deployed, they will need a way to access the information so that they can pay military members accurately. The file formats for these download files have been specified, but the method of access has not yet been determined. If all of the DFAS centers have Internet connectivity, then this would be a prime candidate for adding to the Web interface. Again, this would be a separate menu option, and it would require some type of authentication or user identification, as log files need to be kept to track which services have downloaded. The data to be downloaded is selected based upon criteria such as effective date or location. These selections could be implemented in a Web page as form controls such as list select boxes and text boxes. Clicking the form submit button would initiate the building of the

files and download via FTP. This would be an excellent way to facilitate users who are spread out all over the country accessing a centrally located database, without having to develop and deploy a client server interface.

VI. CONCLUSION

A. SUMMARY

Although the World Wide Web began simply as a means of displaying static data and images, it is rapidly evolving into a powerful environment for delivering dynamic content. With this rapid growth, the union of Web and database technology is inevitable. The capability to access a database through the Web is valuable for both business and entertainment applications. According to Julia Vowler [Ref.4], as of February 1996, only one percent of the world's approximately 300,000 Web sites were connected to a database. Both the number of Web sites in existence and the percentage which access a database have grown over the past year and will undoubtedly grow even more rapidly in the future.

Over the past decade, the information technology industry has spent a great deal of effort on building client/server database applications. These companies have found that developing, deploying, and maintaining client/server applications is inherently difficult due to a variety of factors. Web/database technology offers an alternative to client/server database applications with many advantages. Companies are now beginning to focus their efforts on developing Web/database applications. However, the Web is still an immature technology, and currently, Web implementations are best suited for simpler database applications. Currently, more complex functionality is best implemented as a client/server application. The OHA Web/database interface case study presented in Chapter V is an example of an appropriate application for the Web, as it involves information that is freely available to the public, no updates are necessary, and the tables involved are relatively small.

There are a variety of methods currently available for accessing a database through the Web, and new products and methods are being developed. While the list of options presented in Chapter III is not exhaustive, it provides a good sampling of the most popular methods being used today. Each of the methods described has advantages

and disadvantages. The right solution for any given application ultimately depends upon the characteristics of the application itself. It is also possible to combine components of the various methods. For example, it is possible to use a Java application as a CGI program, or to invoke a CGI program with Oracle's Web Server.

Despite the many advantages of Web/database interfaces, there are many issues yet to be resolved. The Web environment presents new challenges in areas such as speed and performance, concurrency, and security. Many of these challenges are due to the stateless and connectionless nature of the HTTP protocol, as well as the inherently insecure nature of the Internet. While these problems are substantial, the computer industry is focusing a great deal of effort on their solutions. It is expected that many of these problems will be resolved in a future release of the HTTP protocol which will allow for stateful transactions and make it possible to maintain a connection to the database. Security is an issue that is receiving a great deal of attention. Some of the proposed solutions to security problems include authentication of both the client and the server, as well as the use of encryption and firewalls.

B. FUTURE DIRECTIONS

Web technology is undergoing change at an extremely rapid pace. According to Oracle's Magnus Lonroth [Ref. 5], Web product life cycle is reduced to months instead of years. This is due to the great demand to use the Web as a business tool. The combination of fully transactional Web/database interfaces with online commerce has the potential to revolutionize the business world. Once security obstacles have been overcome, companies will be able to offer a wide variety of services over the Web, such as online banking, shopping, booking travel, and much more. In addition, corporate Intranets will allow organizations to build internal networks and make internal applications such as payroll available through the Web without allowing outside users to access these internal functions.

Software vendors are rushing to solve these problems so that such services can be made available. However, it is important that they take the time to develop sound

solutions and deliver quality products if individuals, corporations, and government agencies will be depending upon them for mission critical functions.

The Web has had a tremendous impact upon the computer industry. It is opening up new markets and product categories. Some of the areas that are expected to continue to grow along with the Web include web development tools and languages. Simple HTML coding is being replaced with visual and programmatic interfaces that make Web development easier. In addition, languages such as Java and Active X, which are well-suited for Web development, will become increasingly popular. The object-oriented nature of these languages as well as the desire to store multi-media objects are likely to increase the popularity of object-oriented databases. In anticipation of this, the next releases of Oracle and Informix DBMS's will include object-oriented capabilities. Finally, new types of low cost hardware devices are being developed, such as the Network Computer (NC), which is a low-cost, low maintenance alternative to the Personal Computer (PC) with only simple capabilities such as word processing, e-mail, and Web browsing. This device will have minimal persistent storage; all of the programs will be downloaded through the Internet or a LAN and all of the files will be stored on the network, most likely in a database. This technology is also being combined with the popularity of television to produce a Web-TV product which allows people to access the Web through their television sets. All of this will make Web access more widely available to casual users, and it will become increasingly important to be able to manage all of this data in a secure and well organized manner. The role of database systems becomes extremely important in this scenario. Oracle Corporation's Ray Lane [Ref. 6] states:

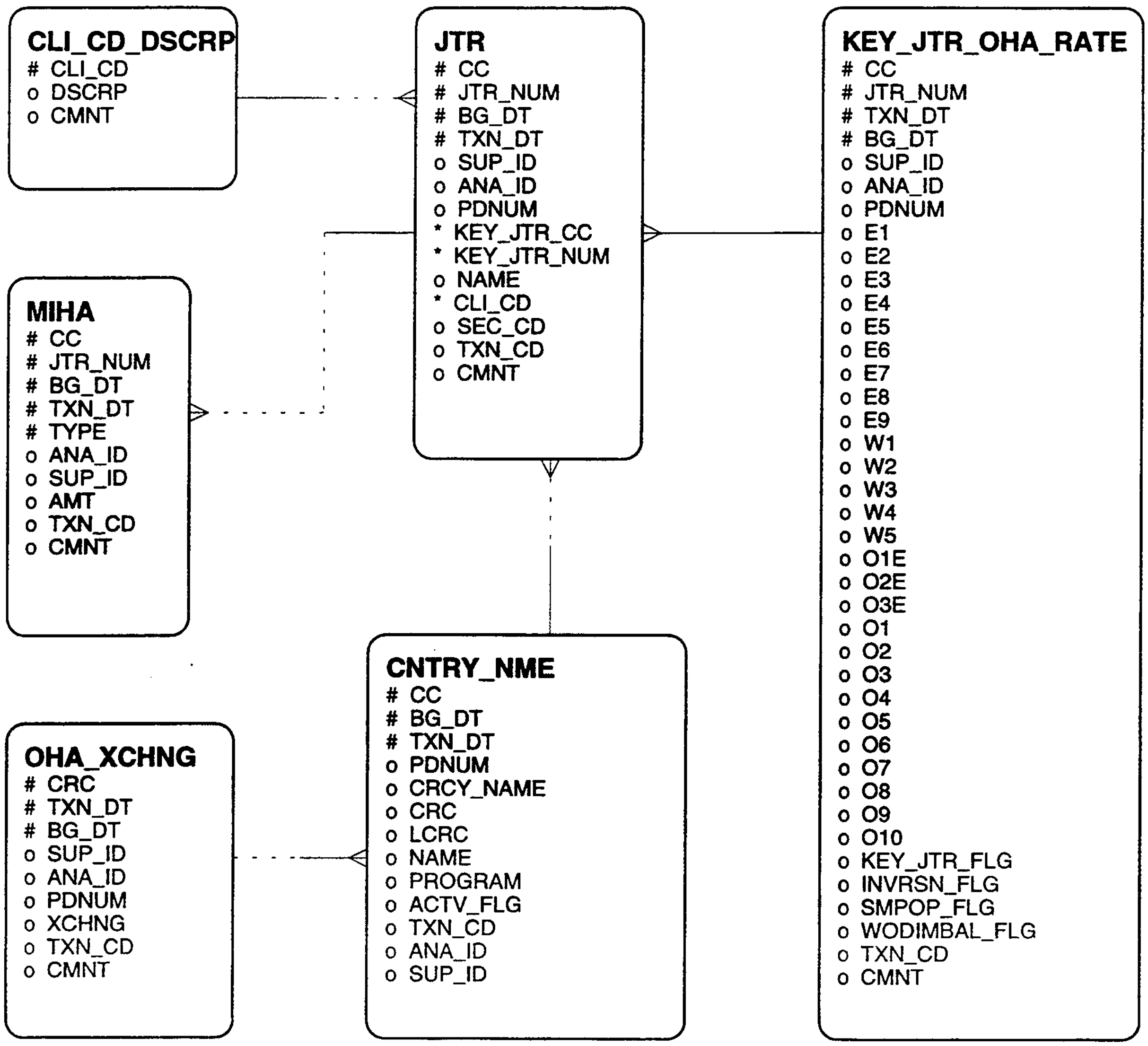
> If a device like the NC allows you more access, then you have many more users and you will have many more databases by definition. The need to have more data, more databases, basically creates more and more back end systems in order to serve all of these devices.

Thus, databases and the ability to access them through the Web are central to this new computing paradigm. Web/database integration is critical to the success of the Network Computing architecture. The combination of the two will not only open up new

applications and capabilities, but it will also make them available to a much wider audience.

APPENDIX A. ENTITY-RELATIONSHIP DIAGRAM FOR THE OHA DATABASE

This Entity-Relationship diagram represents a simplified model of the OHA database described in Chapter V. Only the tables that are relevant to the Web/database interface are included.



Legend

Attributes

#	Unique Identifier
*	Mandatory
o	Optional

Relationships

Optionality

- - - - - - - -	Optional
————————	Mandatory

Cardinality

————————	Single
>———————	Multiple

APPENDIX B. DATABASE SCHEMA FOR THE OHA WEB/DATABASE INTERFACE

This database schema corresponds to the Entity-Relationship diagram shown in Appendix A. It represents a simplified model of the OHA database described in Chapter V. Only the tables that are relevant to the Web/database interface are included.

```
CREATE TABLE CLI_CD_DSCRP
      (CLI_CD              NUMBER(1)
                           CONSTRAINT C_NN_CC_CLI_CD NOT NULL,
       DSCRP               VARCHAR2(30),
       CMNT                LONG,
       CONSTRAINT C_PK_CCD PRIMARY KEY (CLI_CD)
       USING INDEX TABLESPACE OHA_INDEX)
 STORAGE (INITIAL 1K
          NEXT 1K
          PCTINCREASE 0);


CREATE TABLE CNTRY_NME
      (CC                  VARCHAR2(2)
                           CONSTRAINT C_NN_CN_CC NOT NULL,
       BG_DT               DATE
                           CONSTRAINT C_NN_CN_BG_DT NOT NULL,
       TXN_DT              DATE
                           CONSTRAINT C_NN_CN_TXN_DT NOT NULL,
       PDNUM               NUMBER(5),
       CRCY_NAME           VARCHAR2(30),
       CRC                 VARCHAR2(2),
       LCRC                VARCHAR2(2),
       NAME                VARCHAR2(40),
       PROGRAM             NUMBER(1),
       ACTV_FLG            NUMBER(1),
       TXN_CD              NUMBER(2),
       ANA_ID              NUMBER(1),
       SUP_ID              NUMBER(1),
       CONSTRAINT C_PK_CNME PRIMARY KEY (CC,BG_DT,TXN_DT)
       USING INDEX TABLESPACE OHA_INDEX)
STORAGE (INITIAL 5K
         NEXT 5K
         PCTINCREASE 0);
```

```
CREATE TABLE JTR
       (CC                      VARCHAR2(2)
                                CONSTRAINT C_NN_J_CC NOT NULL,
        JTR_NUM                 NUMBER(3)
                                CONSTRAINT C_NN_J_JTR_NUM NOT NULL,
        BG_DT                   DATE
                                CONSTRAINT C_NN_J_BG_DT NOT NULL,
        TXN_DT                  DATE
                                CONSTRAINT C_NN_J_TXN_DT NOT NULL,
        SUP_ID                  NUMBER(1),
        ANA_ID                  NUMBER(1),
        PDNUM                   NUMBER(5),
        KEY_JTR_CC              VARCHAR2(2),
        KEY_JTR_NUM             NUMBER(3),
        NAME                    VARCHAR(40),
        CLI_CD                  NUMBER(1),
        SEC_CD                  NUMBER(1),
        TXN_CD                  NUMBER(2),
        CMNT                    LONG,
       CONSTRAINT C_PK_JTR PRIMARY KEY
       (CC,JTR_NUM,BG_DT,TXN_DT)
       USING INDEX TABLESPACE OHA_INDEX)
STORAGE (INITIAL 20K
         NEXT 20K
         PCTINCREASE 0);


CREATE TABLE OHA_XCHNG
       (CRC                     VARCHAR2(2)
                                CONSTRAINT C_NN_OX_CRC NOT NULL,
        TXN_DT                  DATE
                                CONSTRAINT C_NN_OX_TXN_DT NOT NULL,
        BG_DT                   DATE
                                CONSTRAINT C_NN_OX_BG_DT NOT NULL,
        SUP_ID                  NUMBER(1),
        ANA_ID                  NUMBER(1),
        PDNUM                   NUMBER(5),
        XCHNG                   FLOAT,
        TXN_CD                  NUMBER(2),
        CMNT                    LONG,
       CONSTRAINT C_PK_OXCH PRIMARY KEY (CRC,TXN_DT,BG_DT)
       USING INDEX TABLESPACE OHA_INDEX)
STORAGE (INITIAL 1M
         NEXT 1M
         PCTINCREASE 0);
```

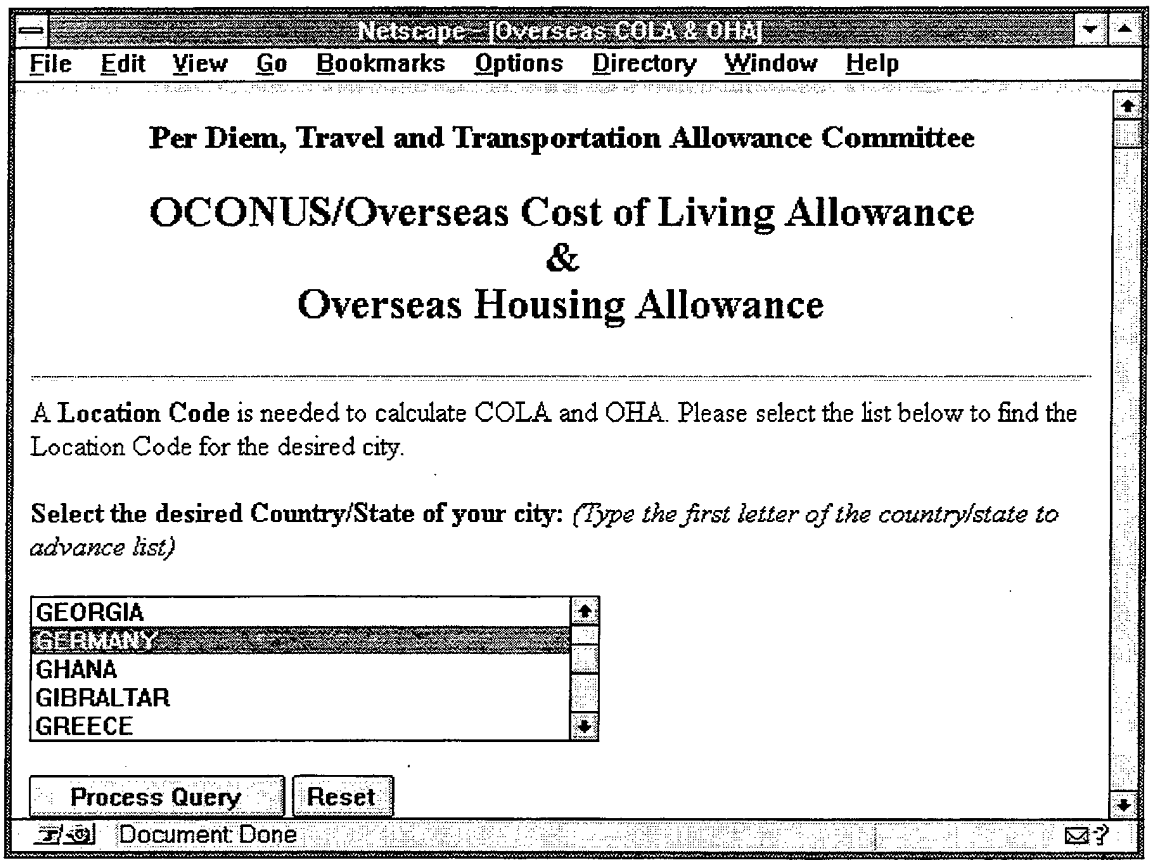
```
CREATE TABLE KEY_JTR_OHA_RATE
      (CC            VARCHAR2(2)
                     CONSTRAINT C_NN_KJOR_CC NOT NULL,
       JTR_NUM       NUMBER(3)
                     CONSTRAINT C_NN_KJOR_JTR_NUM NOT NULL,
       TXN_DT        DATE
                     CONSTRAINT C_NN_KJOR_TXN_DT NOT NULL,
       BG_DT         DATE
                     CONSTRAINT C_NN_KJOR_BG_DT NOT NULL,
       SUP_ID        NUMBER(1),
       ANA_ID        NUMBER(1),
       PDNUM         NUMBER(5),
       E1            FLOAT,
       E2            FLOAT,
       E3            FLOAT,
       E4            FLOAT,
       E5            FLOAT,
       E6            FLOAT,
       E7            FLOAT,
       E8            FLOAT,
       E9            FLOAT,
       W1            FLOAT,
       W2            FLOAT,
       W3            FLOAT,
       W4            FLOAT,
       W5            FLOAT,
       O1E           FLOAT,
       O2E           FLOAT,
       O3E           FLOAT,
       O1            FLOAT,
       O2            FLOAT,
       O3            FLOAT,
       O4            FLOAT,
       O5            FLOAT,
       O6            FLOAT,
       O7            FLOAT,
       O8            FLOAT,
       O9            FLOAT,
       O10           FLOAT,
       KEY_JTR_FLG   NUMBER(1),
       INVRSN_FLG    NUMBER(1),
       SMPOP_FLG     NUMBER(1),
       WODIMBAL_FLG  NUMBER(1),
       TXN_CD        NUMBER(2),
       CMNT          LONG,
```

```
CONSTRAINT C_PK_KJOR PRIMARY KEY    (CC,JTR_NUM,TXN_DT,BG_DT
      USING INDEX TABLESPACE OHA_INDEX)
STORAGE (INITIAL 1M
         NEXT 1M
         PCTINCREASE 0);


CREATE TABLE MIHA
      (CC                 VARCHAR2(2)
                          CONSTRAINT C_NN_M_CC NOT NULL,
       JTR_NUM            NUMBER(3)
                          CONSTRAINT C_NN_M_JTR_NUM NOT NULL,
       BG_DT              DATE
                          CONSTRAINT C_NN_M_BG_DT NOT NULL,
       TXN_DT             DATE
                          CONSTRAINT C_NN_M_TXN_DT NOT NULL,
       TYPE               NUMBER(1)
                          CONSTRAINT C_NN_M_TYPE NOT NULL,
       ANA_ID             NUMBER(1),
       SUP_ID             NUMBER(1),
       AMT                NUMBER(5),
       TXN_CD             NUMBER(2),
       CMNT               LONG,
      CONSTRAINT C_PK_MIHA PRIMARY KEY
      (CC,JTR_NUM,BG_DT,TXN_DT,TYPE)
      USING INDEX TABLESPACE OHA_INDEX)
STORAGE (INITIAL 1K
         NEXT 1K
         PCTINCREASE 0);
```

APPENDIX C. SCREEN VIEWS OF THE EXISTING OHA WEB SITE

This appendix contains the screen views of the existing OHA Web site currently maintained on the DTIC Web server by PDTATAC. Screens that do not fit on one page are shown in printout form.



Country/State: **GERMANY**

Make a note of the LOCATION CODE for your city exactly as shown. Select an option below the list of locations to proceed.

NOTE: The **OTHER** location should be used if the city/installation is not listed

LOCATION-CODE	CITY-NAME	ALLOWANCES	
GM999	ALL OTHER LANDSTATES	COLA	OHA
GM210	ALZEY (RP)	COLA	OHA
GM101	ASCHAFFENBURG (B)	COLA	OHA
GM103	AUGSBURG (INCL LANDSBERG)(B	COLA	OHA
GM301	BABENHAUSEN (H)	COLA	OHA
GM162	BAD AIBLING (B)	COLA	OHA
GM601	BAD KREUZNACH (RP)	COLA	OHA
GM212	BANN (RP)	COLA	OHA
GM603	BAUMHOLDER (INCL BOERFINK) RP)	COLA	OHA
GM201	BERLIN (WESTERN SECTORS)	COLA	OHA
GM511	BIELEFELD (INCL DETMOLD) (NRW)	COLA	OHA
GM109	BINDLACH (B)	COLA	OHA
GM605	BIRKENFELD (RP)	COLA	OHA
GM501	BONN (INCL KOLN/BONN AIRPORT) (NRW)	COLA	OHA
GM503	BORGHOLZHAUSEN (NRW)	COLA	OHA
GM203	BREMEN (INCL BREMERHAVEN AND NORDHOLTZ)	COLA	OHA
GM401	BUECKENBURG (LS)	COLA	OHA
GM307	BUEDINGEN (H)	COLA	OHA
GM507	BURBACH (NRW)	COLA	OHA
GM169	COLOGNE (INCL DELLBRUECK,PORZ-WAHN)(NRW)	COLA	OHA
GM311	DARMSTADT (H)	COLA	OHA
GM663	DEXHEIM (RP)	COLA	OHA
GM071	DONAUESCHINGEN (BW)	COLA	OHA
GM111	ECKSTEIN (RIMBACH) (B)	COLA	OHA
GM609	EINSIDLERHOF (RP)	COLA	OHA
GM165	ERDING (B)	COLA	OHA
GM113	ERLANGEN (B)	COLA	OHA
GM313	ERLANSEE (H)	COLA	OHA
GM079	FELDBERG/SCHWARZWALD (BW)	COLA	OHA
GM317	FRANKFURT AM MAIN (INC RHEIN MAIN AB)(H)	COLA	OHA
GM067	FREIBURG (BW)	COLA	OHA
GM117	FUERTH (B)	COLA	OHA
GM121	GARMISCH (B)	COLA	OHA
GM405	GARTOW (LS)	COLA	OHA
GM531	GEILENKIRCHEN (NRW)	COLA	OHA
GM321	GELNHAUSEN (H)	COLA	OHA
GM123	GIEBELSTADT (B)	COLA	OHA
GM323	GIESSEN (H)	COLA	OHA
GM613	GONSENHEIM (RP)	COLA	OHA
GM224	GOTTINGEN (LS)	COLA	OHA
GM222	GREDING (B)	COLA	OHA
GM017	GROSS ENGSTINGEN (BW)	COLA	OHA
GM327	GROSSAUHEIM (H)	COLA	OHA
GM125	GROSSENGSTIGEN (B)	COLA	OHA
GM523	GUETERSLOH (NRW)	COLA	OHA

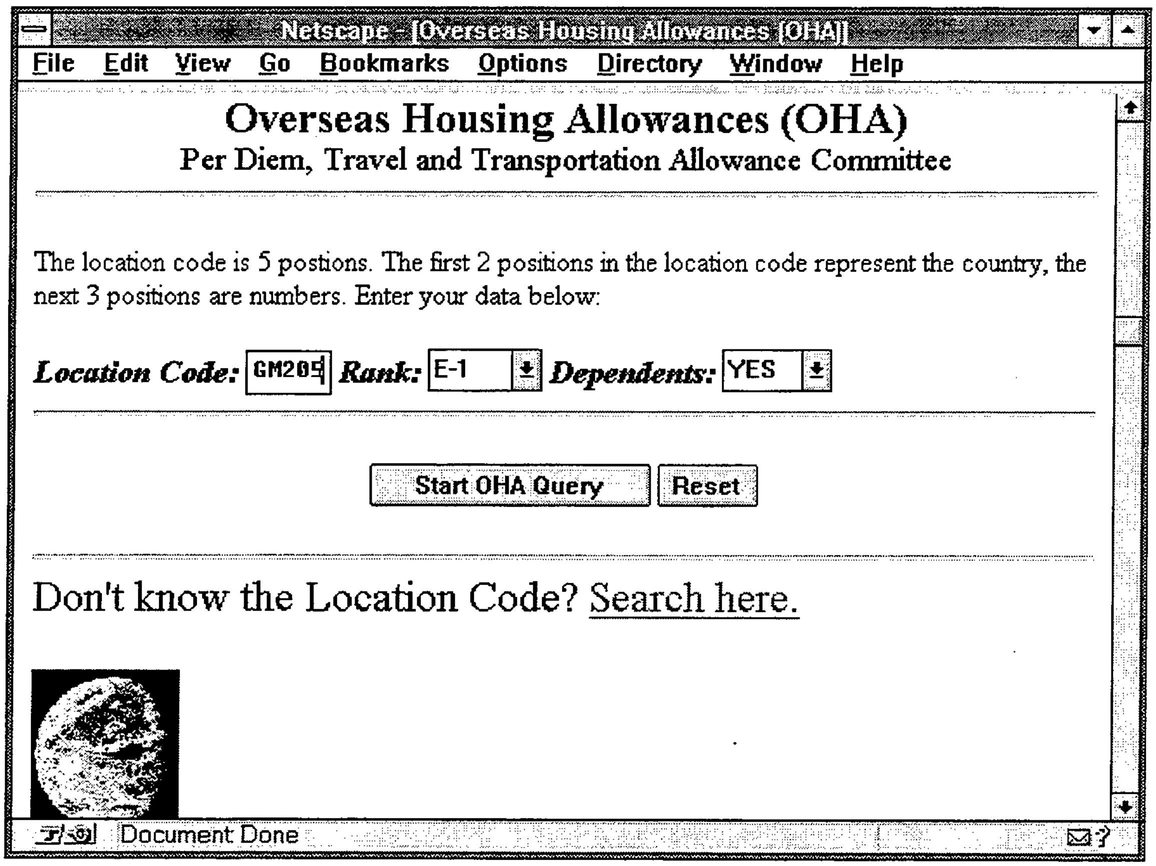
GM205	HAMBURG	COLA	OHA
GM329	HANAU (H)	COLA	OHA
GM407	HANNOVER (INCL WUNSTORF) (LS)	COLA	OHA
GM019	HEIDELBERG (BW)	COLA	OHA
GM409	HELMSTEDT (LS)	COLA	OHA
GM411	HESSICH-OLDENDORF (LS)	COLA	OHA
GM413	HOHNE-BERGEN (LS)	COLA	OHA
GM021	HORMSGRINDE (BW)	COLA	OHA
GM619	IDAR OBERSTEIN (RP)	COLA	OHA
GM815	JENA	COLA	OHA
GM415	JEVER AB (LS)	COLA	OHA
GM621	KAISERSLAUTERN (RP)	COLA	OHA
GM539	KALKAR (NRW)	COLA	OHA
GM025	KALTENBRONN (BW)	COLA	OHA
GM027	KARLSRUHE (INCL ETTLINGEN)(BW)	COLA	OHA
GM221	KIEL (INCL ECKERNFORDE) (SH)	COLA	OHA
GM335	KIRCHGOENS/BUTZBACH (H)	COLA	OHA
GM129	KITZENGEN (INCL WUERZBURG)(B)	COLA	OHA
GM033	KONSTANZ (BW)	COLA	OHA
GM629	LANDSTUHL (RP)	COLA	OHA
GM214	LANGERKOPF (RP)	COLA	OHA
GM039	MANNHEIM (INCL SANDHOFEN) (BW)	COLA	OHA
GM131	MEMMINGEN (B)	COLA	OHA
GM041	MESSETETTEN (BW)	COLA	OHA
GM635	MIESAU (RP)	COLA	OHA
GM339	MUENSTER (H)	COLA	OHA
GM549	MUNCHENGLADBACH (INCL GREFRATH,ETC (NRW)	COLA	OHA
GM639	MUNCHENWEILER (RP)	COLA	OHA
GM133	MUNICH (B) (INCL OBERPFAFFENHOFEN)	COLA	OHA
GM417	MUNSTER-OERTZE (LS)	COLA	OHA
GM641	NEUBRUECKE (RP)	COLA	OHA
GM551	NOERVENICH (NRW)	COLA	OHA
GM137	NURNBERG (B)	COLA	OHA
GM139	OBERAMMERGAU (B)	COLA	OHA
GM219	OBERAMMERGAU MOD	COLA	OHA
GM429	OLDENBERG (LS)	COLA	OHA
GM063	OTHER BADEN-WUERTEMBERG	COLA	OHA
GM153	OTHER BAVARIA	COLA	OHA
GM359	OTHER HESSE	COLA	OHA
GM425	OTHER LOWER SAXONY	COLA	OHA
GM573	OTHER NORTH RHINE WESTPHALIA	COLA	OHA
GM655	OTHER RHINELAND PALATINATE	COLA	OHA
GM207	OTHER SAARLAND	COLA	OHA
GM703	OTHER SCHLESWIG HOLSTEIN	COLA	OHA
GM555	PADERBORN (NRW)	COLA	OHA
GM643	RAMSTEIN (RP)	COLA	OHA
GM723	RENDSBURG (SH)	COLA	OHA
GM199	RHEINBERG (NRW)	COLA	OHA
GM216	RUPPERTSWEILER (RP)	COLA	OHA
GM349	RUSSELSHEIM (H)	COLA	OHA
GM143	SCHWABACH (B)	COLA	OHA
GM645	SEMBACH AB (RP)	COLA	OHA
GM464	SOEGEL (INCL MEPPEN) (LS)	COLA	OHA
GM145	SONTHOFEN (B)	COLA	OHA
GM147	STEIN (B)	COLA	OHA
GM055	STUTTGART MILITARY COMMUNITY (BW)	COLA	OHA

GM353	TREYSA (H)	COLA	OHA
GM059	TUBINGEN (BW)	COLA	OHA
GM599	TWISTEDEN (NRW)	COLA	OHA
GM061	ULM (INCL NEU ULM) (BW)	COLA	OHA
GM649	WACKERHEIM (RP)	COLA	OHA
GM095	WEINGARTEN (BW)	COLA	OHA
GM565	WERL (NRW)	COLA	OHA
GM651	WESTERBURG (RP)	COLA	OHA
GM355	WIESBADEN (H)	COLA	OHA
GM357	WIESBADEN AB (H)	COLA	OHA
GM469	WILHEMSHAVEN (LS)	COLA	OHA
GM653	ZWEIBRUECKEN (INCL KREUZBERG KAS) (RP)	COLA	OHA

FIND Overseas Housing Allowances (OHA)

FIND Cost Of Living Allowance (COLA)

Return to Previous


Netscape - [Overseas Housing Allowances (OHA)]
File Edit View Go Bookmarks Options Directory Window Help
Overseas Housing Allowances (OHA)
Per Diem, Travel and Transportation Allowance Committee
The location code is 5 postions. The first 2 positions in the location code represent the country, the next 3 positions are numbers. Enter your data below:
Location Code: GM205 Rank: E-1 Dependents: YES
Start OHA Query
Reset
Don't know the Location Code? Search here.
Document Done

Netscape - [http://www.dtic.mil/cgi-bin/ohasrch.pl]

File Edit View Go Bookmarks Options Directory Window Help

Here is the information that you requested.

GERMANY

Location: HAMBURG , Location Code: GM205

For an **E-1 with dependents**, the OHA rental ceiling is: **$ 1013.00**

Average utility/recurring maintenance/miscellanous allowance is: **$ 221.00**

Move-In-Housing Allowance is: **$ 419.00**

These allowances are effective: **961016**

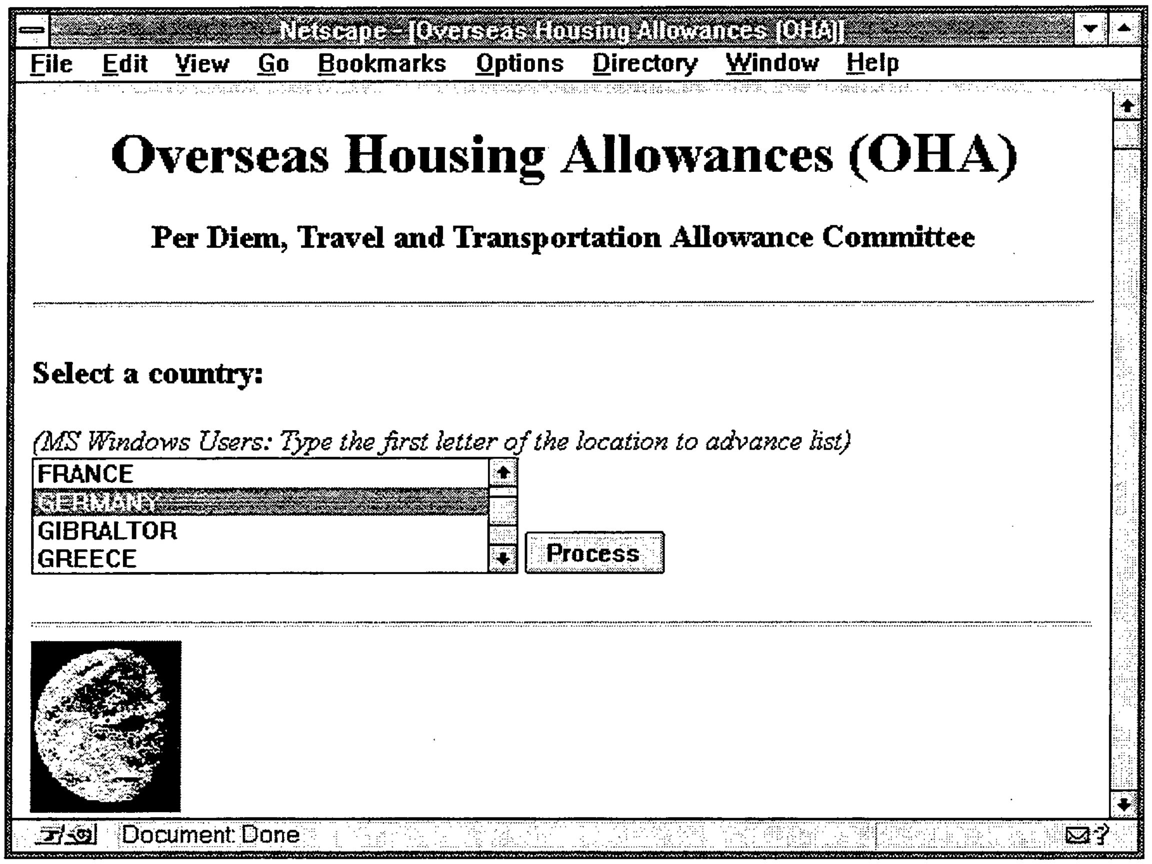
Climate code is: **2.00**

Exchange rate is: **1.48999986590001**

Document: Done

APPENDIX D. SCREEN VIEWS OF THE OHA WEB/DATABASE INTERFACE

This appendix contains the screen views of the prototype OHA Web/database interface developed in Chapter V. Screens that do not fit on one page are shown in printout form.



Netscape - [Overseas Housing Allowances (OHA)]

File Edit View Go Bookmarks Options Directory Window Help

Overseas Housing Allowances (OHA)

Per Diem, Travel and Transportation Allowance Committee

Select a location:

(MS Windows Users: Type the first letter of the location to advance list)

GUETERSLOH (NRW)
HAMBURG
HANAU (H)
HANNOVER (INCL WUNSTORF) (LS)

Process

Document Done

Overseas Housing Allowances (OHA)

Per Diem, Travel and Transportation Allowance Committee

OHA Ceilings for Location: GM205

HAMBURG

Rank	With Dependents	Without Dependents
E-1	$1041	$937
E-2	$1041	$937
E-3	$1041	$937
E-4	$1041	$937
E-5	$1041	$937
E-6	$1641	$1477
E-7	$1776	$1598
E-8	$1910	$1719
E-9	$2066	$1859
W-1	$1721	$1549
W-2	$1721	$1549
W-3	$1910	$1719
W-4	$2103	$1893
W-5	$2103	$1893
O-1E	$1721	$1859
O-2E	$1910	$1719
O-3E	$2103	$1893
O-1	$1721	$1549
O-2	$1721	$1549
O-3	$1910	$1719
O-4	$2103	$1893
O-5	$2103	$1893
O-6	$2103	$1893
O-7	$2103	$1893
O-8	$2103	$1893
O-9	$2103	$1893
O-10	$2103	$1893

Effective date for these allowances: 01-SEP-85

The local currency for this area is: Unknown Currency Name

The exchange rate is: .7042

Moving-In Housing Allowance is: $431

The climate in this area is: Warm

This page was produced by the **Oracle Web Agent** on November 18, 1996 02:03 PM
View PL/SQL source code

APPENDIX E. PL/SQL CODE FOR THE OHA WEB/DATABASE INTERFACE

The following is a listing of the PL/SQL package for the prototype OHA Web/database interface developed in Chapter V.

```
--package specification
PACKAGE OHA_PKG1 IS


        --procedure declarations
        PROCEDURE TSTOHA;
        PROCEDURE TSTOHA2(CC IN VARCHAR2);
        PROCEDURE TSTOHA3 (JTRLOC IN VARCHAR2,
                           IN_CRC IN VARCHAR2,
                           IN_CRCNAME IN VARCHAR2);


END;
```

```
--Package body
PACKAGE BODY OHA_PKG1 IS

   --global variable declaration
   url varchar2(100) := 'http://206.39.184.1';


--This procedure displays retrieves a list of OHA-eligible
countries
--from the database and displays them in a selection list
PROCEDURE TSTOHA IS
        --cursor and local variable declarations
        cursor cntry_cursor is SELECT distinct cntry_nme.cc,
                cntry_nme.name, cntry_nme.crc,
                cntry_nme.crcy_name
                from cntry_nme, key_jtr_oha_rate rate
                where cntry_nme.cc = rate.cc
                order by name;
        selected varchar2(8);
        ctr number := 0;

BEGIN

        --generate HTML tags
        htp.htmlOpen;
        htp.headOpen;
        htp.title( 'Overseas Housing Allowances (OHA)');
        htp.headClose;

        htp.bodyOpen(cbackground => url ||
        '/oha/whtpaper.gif');
        htp.header( 1, 'Overseas Housing Allowances (OHA)',
        'center');
        htp.header(3,
'Per Diem, Travel and Transportation Allowance
Committee','center');
        htp.para;
        htp.line;

        --HTML form tags specify the URL for the next
        --procedure
        htp.formOpen(curl => 'OHA_PKG1.TSTOHA2',cmethod =>
        'POST');
        htp.para;
        htp.header(3,'Select a country: ' );
```

```
        htp.print('<I>(MS Windows Users: Type the first
letter of the location to advance list)</I>');
        htp.nl;

        htp.formSelectOpen('CC',null,4);

        --loop through rows from database and display in a
        --select list.
        --cntry_rec is implicitly declared and cursor is
        --implicitly loaded.
        for cntry_rec in cntry_cursor
        loop
                ctr := ctr + 1;
                --set the first country to be initially
                --selected
                if ctr = 1 then selected := 'SELECTED';
                else selected := null;
                end if;
                --insert the country name into the select
                --list along
                --with associated values to be passed to the
                --next screen
                htp.formSelectOption(cvalue =>
                cntry_rec.name,
                        cselected => selected,
                        cattributes => 'value="' ||
                                       cntry_rec.cc ||
                                       cntry_rec.crc ||
                                       cntry_rec.crcy_name
                                       || '"');
        end loop;
        htp.formSelectClose;

        --Submit button
        htp.formSubmit(cvalue => 'Process');
        htp.formClose;

        htp.line;
        --display icon to return to PDTATAC home page
        htp.anchor('index.html',htf.img(curl => url ||
        '/oha/earthico.gif'));


        htp.para;
        htp.para;
```

```
        htp.para;

        htp.bodyClose;
        htp.htmlClose;


END;
```

```
--This procedure retrieves a list of locations within the
--selected country and displays them in a selection list
PROCEDURE TSTOHA2 (CC in Varchar2) IS
        --cursor and local variable declarations
        in_cc varchar2(2);
        in_crc varchar2(2);
        in_crcy_name varchar2(30);
        cursor loc_cursor is SELECT jtr.cc, jtr.jtr_num,
        name, cli_cd
                from jtr, key_jtr_oha_rate rate
                where jtr.cc = in_cc and
                jtr.cc = rate.cc and jtr.jtr_num =
               rate.jtr_num
                order by name;
        selected varchar2(8);
        ctr number := 0;
BEGIN

        --parse input parameters
        in_cc := substr(CC,1,2);
        in_crc := substr(CC,3,2);
        in_crcy_name := substr(CC,5);

        --generate HTML tags
        htp.htmlOpen;
        htp.headOpen;
        htp.title( 'Overseas Housing Allowances (OHA)');
        htp.headClose;

        htp.bodyOpen(cbackground => url ||
        '/oha/whtpaper.gif');
        htp.header( 1, 'Overseas Housing Allowances (OHA)',
        'center');
        htp.header(3,
        'Per Diem, Travel and Transportation Allowance
        Committee','center');
        htp.para;
        htp.line;

        --HTML form tags specify the URL for the next
        --procedure
        htp.formOpen(curl => 'OHA_PKG1.TSTOHA3',cmethod =>
        'POST');

        htp.para;
```

```
        htp.header(3,'Select a location: ' );
        htp.print('<I>(MS Windows Users: Type the first
letter of the location to advance list)</I>');
        htp.nl;

        htp.formSelectOpen('JTRLOC',null,4);

        --loop through rows from database and display in a
        --select list.
        --loc_rec is implicitly declared and cursor is
        --implicitly loaded.
        for loc_rec in loc_cursor
        loop
                ctr := ctr + 1;
                --set the first country to be initially
                --selected
                if ctr = 1 then selected := 'SELECTED';
                else selected := null;
                end if;
                --insert the location name into the select
                --list along
                --with associated values to be passed to the
                --next screen
                htp.formSelectOption(cvalue => loc_rec.name,
                        cselected => selected,
                        cattributes => 'value="' ||
                                loc_rec.cc ||
                lpad(to_char(loc_rec.jtr_num),3,'0') ||
                                rpad(loc_rec.name,40,' ') ||
                                loc_rec.cli_cd ||
                                '"');
        end loop;
        htp.formSelectClose;

        --Hidden fields to be passed on to the next
        --procedure
        htp.formHidden('IN_CRC',in_crc);
        htp.formHidden('IN_CRCNAME', in_crcy_name);

        --Submit button
        htp.formSubmit(cvalue => 'Process');
        htp.formClose;

        htp.line;
```

```
    --display icon to return to PDTATAC home page
    htp.anchor('index.html',htf.img(curl => url ||
    '/oha/earthico.gif'));

    htp.para;
    htp.para;
    htp.para;
    htp.bodyClose;
    htp.htmlClose;
END;
```

```
--This procedure retrieves OHA rates and associated
--information from the database based upon the country
--and location selections and displays them
--in a table format.
PROCEDURE TSTOHA3 (JTRLOC IN VARCHAR2,
                   IN_CRC IN VARCHAR2,
                   IN_CRCNAME IN VARCHAR2) IS
--cursor and local variable declarations
in_cc varchar2(2);
in_jtr varchar2(3);
in_name varchar2(40);
in_climate varchar2(1);

cursor loc_cursor is SELECT * FROM key_jtr_oha_rate
        WHERE  cc = in_cc and jtr_num = in_jtr;
--explicit declaration for loc_rec using %rowtype attribute
loc_rec  loc_cursor%rowtype;
cursor cli_cursor is SELECT dscrp FROM cli_cd_dscrp
        WHERE  cli_cd = in_climate;
cli_dscrp varchar2(30);
cursor xch_cursor is SELECT xchng FROM oha_xchng
        WHERE  crc = IN_CRC;
xch_rec  xch_cursor%rowtype;
cursor miha_cursor is SELECT sum(amt) FROM miha
        WHERE  cc = IN_CC and jtr_num = in_jtr;
miha_amt  number;

BEGIN

        --parse input parameters
        in_cc := substr(JTRLOC,1,2);
        in_jtr := substr(JTRLOC,3,3);
        in_name := substr(JTRLOC,6,40);
        in_climate := substr(JTRLOC,46,1);

        --explicitly retrieve data into the cursors
        open loc_cursor;
        fetch loc_cursor into loc_rec;
        close loc_cursor;

        open cli_cursor;
        fetch cli_cursor into cli_dscrp;
        close cli_cursor;
```

```
        open xch_cursor;
        fetch xch_cursor into xch_rec;
        close xch_cursor;

        open miha_cursor;
        fetch miha_cursor into miha_amt;
        close miha_cursor;

        --generate HTML tags
        htp.htmlOpen;
        htp.headOpen;
        htp.title( 'Overseas Housing Allowances (OHA)');
        htp.headClose;
        htp.bodyOpen(cbackground => url ||
        '/oha/whtpaper.gif');
        htp.header( 1, 'Overseas Housing Allowances (OHA)',
        'center');
        htp.header(3, 'Per Diem, Travel and Transportation
Allowance Committee','center');
        htp.para;
        htp.line;
        htp.para;

        htp.centerOpen;
        htp.header( 3, 'OHA Ceilings for Location: '|| in_cc
        || in_jtr, 'center');
        htp.header( 3, in_name, 'center');
        htp.nl;

        --HTML table tags
        htp.tableOpen(cattributes => 'border=2 width=70%' );

        htp.tableRowOpen;
        htp.tableHeader('Rank', cattributes =>
        ' width=10%');
        htp.tableHeader('With Dependents', cattributes =>
        ' width=30%');
        htp.tableHeader('Without Dependents', cattributes =>
        ' width=30%');
        htp.tableRowClose;

        --display OHA rates in an HTML table by dependency
        -- status and rank.
        --Without dependents rates are calculated as 90% of
        -- with dependents rates.
```

```
htp.tableRowOpen;
htp.tableData( htf.strong('E-1'), 'middle');
htp.tableData('$' || loc_rec.E1, 'middle');
htp.tableData('$' || round(loc_rec.E1*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('E-2'), 'middle');
htp.tableData('$' || loc_rec.E2, 'middle');
htp.tableData('$' || round(loc_rec.E2*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('E-3'), 'middle');
htp.tableData('$' || loc_rec.E3, 'middle');
htp.tableData('$' || round(loc_rec.E3*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('E-4'), 'middle');
htp.tableData('$' || loc_rec.E4,'middle');
htp.tableData('$' || round(loc_rec.E4*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('E-5'), 'middle');
htp.tableData('$' || loc_rec.E5,'middle');
htp.tableData('$' || round(loc_rec.E5*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('E-6'), 'middle');
htp.tableData('$' || loc_rec.E6,'middle');
htp.tableData('$' || round(loc_rec.E6*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('E-7'), 'middle');
htp.tableData('$' || loc_rec.E7,'middle');
htp.tableData('$' || round(loc_rec.E7*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('E-8'), 'middle');
htp.tableData('$' || loc_rec.E8,'middle');
htp.tableData('$' || round(loc_rec.E8*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('E-9'), 'middle');
htp.tableData('$' || loc_rec.E9,'middle');
htp.tableData('$' || round(loc_rec.E9*.9),'middle');
htp.tableRowClose;
```

```
htp.tableRowOpen;
htp.tableData( htf.strong('W-1'), 'middle');
htp.tableData('$' || loc_rec.W1,'middle');
htp.tableData('$' || round(loc_rec.W1*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('W-2'), 'middle');
htp.tableData('$' || loc_rec.W2,'middle');
htp.tableData('$' || round(loc_rec.W2*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('W-3'), 'middle');
htp.tableData('$' || loc_rec.W3,'middle');
htp.tableData('$' || round(loc_rec.W3*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('W-4'), 'middle');
htp.tableData('$' || loc_rec.W4,'middle');
htp.tableData('$' || round(loc_rec.W4*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('W-5'), 'middle');
htp.tableData('$' || loc_rec.W5,'middle');
htp.tableData('$' || round(loc_rec.W5*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-1E'), 'middle');
htp.tableData('$'|| loc_rec.O1E,'middle');
htp.tableData('$'|| round(loc_rec.E9*.9), 'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-2E'), 'middle');
htp.tableData('$'|| loc_rec.O2E,'middle');
htp.tableData('$'|| round(loc_rec.O2E*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-3E'), 'middle');
htp.tableData('$'|| loc_rec.O3E,'middle');
htp.tableData('$'|| round(loc_rec.O3E*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-1'), 'middle');
htp.tableData('$' || loc_rec.O1,'middle');
htp.tableData('$' || round(loc_rec.O1*.9),'middle');
htp.tableRowClose;
```

```
htp.tableRowOpen;
htp.tableData( htf.strong('O-2'), 'middle');
htp.tableData('$' || loc_rec.O2,'middle');
htp.tableData('$' || round(loc_rec.O2*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-3'), 'middle');
htp.tableData('$' || loc_rec.O3,'middle');
htp.tableData('$' || round(loc_rec.O3*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-4'), 'middle');
htp.tableData('$' || loc_rec.O4,'middle');
htp.tableData('$' || round(loc_rec.O4*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-5'), 'middle');
htp.tableData('$' || loc_rec.O5,'middle');
htp.tableData('$' || round(loc_rec.O5*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-6'), 'middle');
htp.tableData('$' || loc_rec.O6,'middle');
htp.tableData('$' || round(loc_rec.O6*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-7'), 'middle');
htp.tableData('$' || loc_rec.O7,'middle');
htp.tableData('$' || round(loc_rec.O7*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-8'), 'middle');
htp.tableData('$' || loc_rec.O8,'middle');
htp.tableData('$' || round(loc_rec.O8*.9),middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-9'), 'middle');
htp.tableData('$' || loc_rec.O9,'middle');
htp.tableData('$' || round(loc_rec.O9*.9),'middle');
htp.tableRowClose;
htp.tableRowOpen;
htp.tableData( htf.strong('O-10'), 'middle');
htp.tableData('$'|| loc_rec.O10,'middle');
htp.tableData('$'|| round(loc_rec.O10*.9),'middle');
htp.tableRowClose;
```

```
        htp.tableClose;
        htp.para;

        htp.centerClose;

        --Display associated information
        htp.Header(4,'Effective date for these allowances: '
        || to_char(loc_rec.bg_dt));
        htp.Header(4,'The local currency for this area is: '
        || IN_CRCNAME);
        htp.Header(4,'The exchange rate is:  ' ||
        to_char(xch_rec.xchng));
        htp.Header(4,'Moving-In Housing Allowance is: $' ||
        to_char(miha_amt));
        htp.Header(4,'The climate in this area is: ' ||
        cli_dscrp);
        htp.nl;

        --this statement displays the source code of this
        -- package... for debugging purposes only
        owa_util.signature('OHA_PKG1.tstoha2');
        htp.line;



        --display icon to return to PDTATAC home page
        htp.anchor('index.html',htf.img(curl => url ||
        '/oha/earthico.gif'));


        htp.para;
        htp.para;
        htp.para;

        htp.bodyClose;
        htp.htmlClose;
END;


END;
```

APPENDIX F. GENERATED HTML FOR THE OHA WEB/DATABASE INTERFACE

The following is a listing of the HTML generated by the PL/SQL package shown in Appendix E.

```
<HTML>
<HEAD>
<TITLE>Overseas Housing Allowances (OHA)</TITLE>
</HEAD>
<BODY BACKGROUND="http://206.39.184.1/oha/whtpaper.gif">
<H1 ALIGN="center">Overseas Housing Allowances (OHA)</H1>
<H3 ALIGN="center">Per Diem, Travel and Transportation
Allowance Committee</H3>
<P>
<HR>
<FORM ACTION="OHA_PKG1.TSTOHA2" METHOD="POST">
<P>
<H3>Select a country: </H3>
<I>(MS Windows Users: Type the first letter of the location
to advance list)</I>
<BR>
<SELECT NAME="CC" SIZE="4">
<OPTION SELECTED value="AQAQUnknown Currency Name">AMERICAN
SAMOA
<OPTION value="ACACUnknown Currency Name">ANTIGUA & BARBUDA
<OPTION value="ARARUnknown Currency Name">ARGENTINA
<OPTION value="ASASUnknown Currency Name">AUSTRALIA
<OPTION value="AUAUUnknown Currency Name">AUSTRIA
<OPTION value="BFBFUnknown Currency Name">BAHAMAS  THE
<OPTION value="BABAUnknown Currency Name">BAHRAIN
<OPTION value="BBBBUnknown Currency Name">BARBADOS
<OPTION value="BEBEUnknown Currency Name">BELGIUM
<OPTION value="BHBHUnknown Currency Name">BELIZE (BRITISH
HONDURAS)
<OPTION value="BLBLUnknown Currency Name">BOLIVIA
<OPTION value="BRBRUnknown Currency Name">BRAZIL
<OPTION value="CACAUnknown Currency Name">CANADA
<OPTION value="CICIUnknown Currency Name">CHILE
<OPTION value="CHCHUnknown Currency Name">CHINA  COMMUNIST
```

```
<OPTION value="COCOUnknown Currency Name">COLOMBIA
<OPTION value="CSCSUnknown Currency Name">COSTA RICA
<OPTION value="EZEZUnknown Currency Name">CZECH REPUBLIC
<OPTION value="DADAUnknown Currency Name">DENMARK
<OPTION value="DRDRUnknown Currency Name">DOMINICAN REPUBLIC
<OPTION value="ECECUnknown Currency Name">ECUADOR
<OPTION value="EGEGUnknown Currency Name">EGYPT
<OPTION value="FMFMUnknown Currency Name">FED STATES OF
MICRONESIA
<OPTION value="FJFJUnknown Currency Name">FIJI
<OPTION value="FIFIUnknown Currency Name">FINLAND
<OPTION value="FRFRUnknown Currency Name">FRANCE
<OPTION value="GMGMUnknown Currency Name">GERMANY
<OPTION value="GIGIUnknown Currency Name">GIBRALTOR
<OPTION value="GRGRUnknown Currency Name">GREECE
<OPTION value="GQGQUnknown Currency Name">GUAM
<OPTION value="HAHAUnknown Currency Name">HAITI
<OPTION value="HOHOUnknown Currency Name">HONDURAS
<OPTION value="HUHUUnknown Currency Name">HUNGARY
<OPTION value="IDIDUnknown Currency Name">INDONESIA
<OPTION value="EIEIUnknown Currency Name">IRELAND
<OPTION value="ISISUnknown Currency Name">ISRAEL
<OPTION value="ITITUnknown Currency Name">ITALY
<OPTION value="JMJMUnknown Currency Name">JAMAICA
<OPTION value="JAJAUnknown Currency Name">JAPAN
<OPTION value="JSJSUnknown Currency Name">JERUSALEM
<OPTION value="KEKEUnknown Currency Name">KENYA
<OPTION value="KSKSUnknown Currency Name">KOREA (SOUTH)
<OPTION value="LULUUnknown Currency Name">LUXEMBOURG
<OPTION value="MYMYUnknown Currency Name">MALAYSIA
<OPTION value="RMRMUnknown Currency Name">MARSHALL ISLANDS
<OPTION value="MXMXUnknown Currency Name">MEXICO
<OPTION value="MHMHUnknown Currency Name">MONTSERRAT
<OPTION value="MOMOUnknown Currency Name">MOROCCO
<OPTION value="NLNLUnknown Currency Name">NETHERLANDS
<OPTION value="NZNZUnknown Currency Name">NEW ZEALAND
<OPTION value="NONOUnknown Currency Name">NORWAY
<OPTION value="PKPKUnknown Currency Name">PAKISTAN
<OPTION value="PMPMUnknown Currency Name">PANAMA
<OPTION value="PPPPUnknown Currency Name">PAPUA NEW GUINEA
<OPTION value="PEPEUnknown Currency Name">PERU
<OPTION value="RPRPUnknown Currency Name">PHILLIPINES
<OPTION value="PLPLUnknown Currency Name">POLAND
<OPTION value="POPOUnknown Currency Name">PORTUGAL
<OPTION value="RQRQUnknown Currency Name">PUERTO RICO
```

```
<OPTION value="RWRWUnknown Currency Name">RWANDA
<OPTION value="SNSNUnknown Currency Name">SINGAPORE
<OPTION value="SPSPUnknown Currency Name">SPAIN
<OPTION value="STUnknown Currency Name">ST. LUCIA
<OPTION value="SWSWUnknown Currency Name">SWEDEN
<OPTION value="SZSZUnknown Currency Name">SWITZERLAND
<OPTION value="THTHUnknown Currency Name">THAILAND
<OPTION value="TSTSUnknown Currency Name">TUNISIA
<OPTION value="TUTUUnknown Currency Name">TURKEY
<OPTION value="TCUnknown Currency Name">UNITED ARAB EMIRATES
<OPTION value="UKUKUnknown Currency Name">UNITED KINGDOM
<OPTION value="VEVEUnknown Currency Name">VENEZUELA
<OPTION value="VMUnknown Currency Name">VIETNAM (FORMERLY VN
& VS)
<OPTION value="VQVQUnknown Currency Name">VIRGIN ISLANDS
<OPTION value="ZIZIUnknown Currency Name">ZIMBABWE (FORMERLY
RHODESIA)
</SELECT>
<INPUT TYPE="submit" VALUE="Process">
</FORM>
<HR>
<A HREF="index.html"><IMG
SRC="http://206.39.184.1/oha/earthico.gif"></A>
<P>
<P>
<P>
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>Overseas Housing Allowances (OHA)</TITLE>
</HEAD>
<BODY BACKGROUND="http://206.39.184.1/oha/whtpaper.gif">
<H1 ALIGN="center">Overseas Housing Allowances (OHA)</H1>
<H3 ALIGN="center">Per Diem, Travel and Transportation
Allowance Committee</H3>
<P>
<HR>
<FORM ACTION="OHA_PKG1.TSTOHA3" METHOD="POST">
<P>
<H3>Select a location: </H3>
<I>(MS Windows Users: Type the first letter of the location
to advance list)</I>
<BR>
<SELECT NAME="JTRLOC" SIZE="4">
<OPTION SELECTED value="GM999ALL OTHER LANDSTATES
2">ALL OTHER LANDSTATES
<OPTION value="GM210ALZEY (RP)
2">ALZEY (RP)
<OPTION value="GM101ASCHAFFENBURG (B)
2">ASCHAFFENBURG (B)
<OPTION value="GM103AUGSBURG (INCL LANDSBERG)(B)
2">AUGSBURG (INCL LANDSBERG)(B)
<OPTION value="GM301BABENHAUSEN (H)
2">BABENHAUSEN (H)
<OPTION value="GM162BAD AIBLING (B)
2">BAD AIBLING (B)
<OPTION value="GM601BAD KREUZNACH (RP)
2">BAD KREUZNACH (RP)
<OPTION value="GM212BANN (RP)
2">BANN (RP)
<OPTION value="GM603BAUMHOLDER (INCL BOERFINK) (RP)
2">BAUMHOLDER (INCL BOERFINK) (RP)
<OPTION value="GM201BERLIN (WESTERN SECTORS)
2">BERLIN (WESTERN SECTORS)
<OPTION value="GM511BIELEFELD (INCL DETMOLD) (NRW)
2">BIELEFELD (INCL DETMOLD) (NRW)
<OPTION value="GM109BINDLACH (B)
2">BINDLACH (B)
<OPTION value="GM605BIRKENFELD (RP)
2">BIRKENFELD (RP)
<OPTION value="GM501BONN (INCL KOLN/BONN AIRPORT) (NRW)
2">BONN (INCL KOLN/BONN AIRPORT) (NRW)
```

```
<OPTION value="GM503BORGHOLZHAUSEN (NRW)
2">BORGHOLZHAUSEN (NRW)
<OPTION value="GM203BREMEN (INCL BREMERHAVEN AND NORDHOLTZ)
2">BREMEN (INCL BREMERHAVEN AND NORDHOLTZ)
<OPTION value="GM401BUECKENBURG (LS)
2">BUECKENBURG (LS)
<OPTION value="GM307BUEDINGEN (H)
2">BUEDINGEN (H)
<OPTION value="GM507BURBACH (NRW)
2">BURBACH (NRW)
<OPTION value="GM169COLOGNE (INCL DELLBRUECK,PORZ-
WAHN)(NRW)2">COLOGNE (INCL DELLBRUECK,PORZ-WAHN)(NRW)
<OPTION value="GM311DARMSTADT (H)
2">DARMSTADT (H)
<OPTION value="GM663DEXHEIM (RP)
2">DEXHEIM (RP)
<OPTION value="GM071DONAUESCHINGEN (BW)
2">DONAUESCHINGEN (BW)
<OPTION value="GM111ECKSTEIN (RIMBACH) (B)
2">ECKSTEIN (RIMBACH) (B)
<OPTION value="GM609EINSIDLERHOF (RP)
2">EINSIDLERHOF (RP)
<OPTION value="GM165ERDING (B)
2">ERDING (B)
<OPTION value="GM113ERLANGEN (B)
2">ERLANGEN (B)
<OPTION value="GM313ERLANSEE (H)
2">ERLANSEE (H)
<OPTION value="GM079FELDBERG/SCHWARZWALD (BW)
2">FELDBERG/SCHWARZWALD (BW)
<OPTION value="GM317FRANKFURT AM MAIN (INC RHEIN MAIN
AB)(H)2">FRANKFURT AM MAIN (INC RHEIN MAIN AB)(H)
<OPTION value="GM067FREIBURG (BW)
2">FREIBURG (BW)
<OPTION value="GM117FUERTH (B)
2">FUERTH (B)
<OPTION value="GM121GARMISCH (B)
2">GARMISCH (B)
<OPTION value="GM405GARTOW (LS)
2">GARTOW (LS)
<OPTION value="GM531GEILENKIRCHEN (NRW)
2">GEILENKIRCHEN (NRW)
<OPTION value="GM321GELNHAUSEN (H)
2">GELNHAUSEN (H)
```

```
<OPTION value="GM123GIEBELSTADT (B)
2">GIEBELSTADT (B)
<OPTION value="GM323GIESSEN (H)
2">GIESSEN (H)
<OPTION value="GM613GONSENHEIM (RP)
2">GONSENHEIM (RP)
<OPTION value="GM224GOTTINGEN (LS)
2">GOTTINGEN (LS)
<OPTION value="GM222GREDING (B)
2">GREDING (B)
<OPTION value="GM017GROSS ENGSTINGEN (BW)
2">GROSS ENGSTINGEN (BW)
<OPTION value="GM327GROSSAUHEIM (H)
2">GROSSAUHEIM (H)
<OPTION value="GM125GROSSENGSTIGEN (B)
2">GROSSENGSTIGEN (B)
<OPTION value="GM523GUETERSLOH (NRW)
2">GUETERSLOH (NRW)
<OPTION value="GM205HAMBURG
2">HAMBURG
<OPTION value="GM329HANAU (H)
2">HANAU (H)
<OPTION value="GM407HANNOVER (INCL WUNSTORF) (LS)
2">HANNOVER (INCL WUNSTORF) (LS)
<OPTION value="GM019HEIDELBERG (BW)
2">HEIDELBERG (BW)
<OPTION value="GM409HELMSTEDT (LS)
2">HELMSTEDT (LS)
<OPTION value="GM411HESSICH-OLDENDORF (LS)
2">HESSICH-OLDENDORF (LS)
<OPTION value="GM413HOHNE-BERGEN (LS)
2">HOHNE-BERGEN (LS)
<OPTION value="GM021HORMSGRINDE (BW)
2">HORMSGRINDE (BW)
<OPTION value="GM619IDAR OBERSTEIN (RP)
2">IDAR OBERSTEIN (RP)
<OPTION value="GM815JENA
2">JENA
<OPTION value="GM415JEVER AB (LS)
2">JEVER AB (LS)
<OPTION value="GM621KAISERSLAUTERN (RP)
2">KAISERSLAUTERN (RP)
<OPTION value="GM539KALKAR (NRW)
2">KALKAR (NRW)
```

```
<OPTION value="GM025KALTENBRONN (BW)
2">KALTENBRONN (BW)
<OPTION value="GM027KARLSRUHE (INCL ETTLINGEN)(BW)
2">KARLSRUHE (INCL ETTLINGEN)(BW)
<OPTION value="GM221KIEL (INCL ECKERNFORDE) (SH)
2">KIEL (INCL ECKERNFORDE) (SH)
<OPTION value="GM335KIRCHGOENS/BUTZBACH (H)
2">KIRCHGOENS/BUTZBACH (H)
<OPTION value="GM129KITZENGEN (INCL WUERZBURG)(B)
2">KITZENGEN (INCL WUERZBURG)(B)
<OPTION value="GM033KONSTANZ (BW)
2">KONSTANZ (BW)
<OPTION value="GM629LANDSTUHL (RP)
2">LANDSTUHL (RP)
<OPTION value="GM214LANGERKOPF (RP)
2">LANGERKOPF (RP)
<OPTION value="GM039MANNHEIM (INCL SANDHOFEN) (BW)
2">MANNHEIM (INCL SANDHOFEN) (BW)
<OPTION value="GM131MEMMINGEN (B)
2">MEMMINGEN (B)
<OPTION value="GM041MESSETETTEN (BW)
2">MESSETETTEN (BW)
<OPTION value="GM635MIESAU (RP)
2">MIESAU (RP)
<OPTION value="GM339MUENSTER (H)
2">MUENSTER (H)
<OPTION value="GM549MUNCHENGLADBACH (INCL GREFRATH,ETC
(NRW)2">MUNCHENGLADBACH (INCL GREFRATH,ETC (NRW)
<OPTION value="GM639MUNCHENWEILER (RP)
2">MUNCHENWEILER (RP)
<OPTION value="GM133MUNICH (B) (INCL OBERPFAFFENHOFEN)
2">MUNICH (B) (INCL OBERPFAFFENHOFEN)
<OPTION value="GM417MUNSTER-OERTZE (LS)
2">MUNSTER-OERTZE (LS)
<OPTION value="GM641NEUBRUECKE (RP)
2">NEUBRUECKE (RP)
<OPTION value="GM551NOERVENICH (NRW)
2">NOERVENICH (NRW)
<OPTION value="GM137NURNBERG (B)
2">NURNBERG (B)
<OPTION value="GM139OBERAMMERGAU (B)
2">OBERAMMERGAU (B)
<OPTION value="GM219OBERAMMERGAU MOD
2">OBERAMMERGAU MOD
```

```
<OPTION value="GM429OLDENBERG (LS)
2">OLDENBERG (LS)
<OPTION value="GM063OTHER BADEN-WUERTEMBERG
2">OTHER BADEN-WUERTEMBERG
<OPTION value="GM153OTHER BAVARIA
2">OTHER BAVARIA
<OPTION value="GM359OTHER HESSE
2">OTHER HESSE
<OPTION value="GM425OTHER LOWER SAXONY
2">OTHER LOWER SAXONY
<OPTION value="GM573OTHER NORTH RHINE WESTPHALIA
2">OTHER NORTH RHINE WESTPHALIA
<OPTION value="GM655OTHER RHINELAND PALATINATE
2">OTHER RHINELAND PALATINATE
<OPTION value="GM207OTHER SAARLAND
2">OTHER SAARLAND
<OPTION value="GM703OTHER SCHLESWIG HOLSTEIN
2">OTHER SCHLESWIG HOLSTEIN
<OPTION value="GM555PADERBORN (NRW)
2">PADERBORN (NRW)
<OPTION value="GM643RAMSTEIN (RP)
2">RAMSTEIN (RP)
<OPTION value="GM723RENDSBURG (SH)
2">RENDSBURG (SH)
<OPTION value="GM199RHEINBERG (NRW)
2">RHEINBERG (NRW)
<OPTION value="GM216RUPPERTSWEILER (RP)
2">RUPPERTSWEILER (RP)
<OPTION value="GM349RUSSELSHEIM (H)
2">RUSSELSHEIM (H)
<OPTION value="GM143SCHWABACH (B)
2">SCHWABACH (B)
<OPTION value="GM645SEMBACH AB (RP)
2">SEMBACH AB (RP)
<OPTION value="GM464SOEGEL (INCL MEPPEN) (LS)
2">SOEGEL (INCL MEPPEN) (LS)
<OPTION value="GM145SONTHOFEN (B)
2">SONTHOFEN (B)
<OPTION value="GM147STEIN (B)
2">STEIN (B)
<OPTION value="GM055STUTTGART MILITARY COMMUNITY (BW)
2">STUTTGART MILITARY COMMUNITY (BW)
<OPTION value="GM353TREYSA (H)
2">TREYSA (H)
```

```
<OPTION value="GM059TUBINGEN (BW)
2">TUBINGEN (BW)
<OPTION value="GM599TWISTEDEN (NRW)
2">TWISTEDEN (NRW)
<OPTION value="GM061ULM (INCL NEU ULM) (BW)
2">ULM (INCL NEU ULM) (BW)
<OPTION value="GM649WACKERHEIM (RP)
2">WACKERHEIM (RP)
<OPTION value="GM095WEINGARTEN (BW)
2">WEINGARTEN (BW)
<OPTION value="GM565WERL (NRW)
2">WERL (NRW)
<OPTION value="GM651WESTERBURG (RP)
2">WESTERBURG (RP)
<OPTION value="GM355WIESBADEN (H)
2">WIESBADEN (H)
<OPTION value="GM357WIESBADEN AB (H)
2">WIESBADEN AB (H)
<OPTION value="GM469WILHEMSHAVEN (LS)
2">WILHEMSHAVEN (LS)
<OPTION value="GM653ZWEIBRUECKEN (INCL KREUZBERG KAS) (RP)
2">ZWEIBRUECKEN (INCL KREUZBERG KAS) (RP)
</SELECT>
<INPUT TYPE="hidden" NAME="IN_CRC" VALUE="GM">
<INPUT TYPE="hidden" NAME="IN_CRCNAME" VALUE="Unknown
Currency Name">
<INPUT TYPE="submit" VALUE="Process">
</FORM>
<HR>
<A HREF="index.html"><IMG
SRC="http://206.39.184.1/oha/earthico.gif"></A>
<P>
<P>
<P>
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>Overseas Housing Allowances (OHA)</TITLE>
</HEAD>
<BODY BACKGROUND="http://206.39.184.1/oha/whtpaper.gif">
<H1 ALIGN="center">Overseas Housing Allowances (OHA)</H1>
<H3 ALIGN="center">Per Diem, Travel and Transportation
Allowance Committee</H3>
<P>
<HR>
<P>
<CENTER>
<H3 ALIGN="center">OHA Ceilings for Location: GM205</H3>
<H3 ALIGN="center">HAMBURG
</H3>
<BR>
<TABLE  border=2 width=70%>
<TR>
<TH  width=10%>Rank</TH>
<TH  width=30%>With Dependents</TH>
<TH  width=30%>Without Dependents</TH>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-1</STRONG></TD>
<TD ALIGN="middle">$1041</TD>
<TD ALIGN="middle">$937</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-2</STRONG></TD>
<TD ALIGN="middle">$1041</TD>
<TD ALIGN="middle">$937</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-3</STRONG></TD>
<TD ALIGN="middle">$1041</TD>
<TD ALIGN="middle">$937</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-4</STRONG></TD>
<TD ALIGN="middle">$1041</TD>
<TD ALIGN="middle">$937</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-5</STRONG></TD>
<TD ALIGN="middle">$1041</TD>
```

```
<TD ALIGN="middle">$937</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-6</STRONG></TD>
<TD ALIGN="middle">$1641</TD>
<TD ALIGN="middle">$1477</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-7</STRONG></TD>
<TD ALIGN="middle">$1776</TD>
<TD ALIGN="middle">$1598</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-8</STRONG></TD>
<TD ALIGN="middle">$1910</TD>
<TD ALIGN="middle">$1719</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>E-9</STRONG></TD>
<TD ALIGN="middle">$2066</TD>
<TD ALIGN="middle">$1859</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>W-1</STRONG></TD>
<TD ALIGN="middle">$1721</TD>
<TD ALIGN="middle">$1549</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>W-2</STRONG></TD>
<TD ALIGN="middle">$1721</TD>
<TD ALIGN="middle">$1549</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>W-3</STRONG></TD>
<TD ALIGN="middle">$1910</TD>
<TD ALIGN="middle">$1719</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>W-4</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>W-5</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
```

```
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-1E</STRONG></TD>
<TD ALIGN="middle">$1721</TD>
<TD ALIGN="middle">$1859</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-2E</STRONG></TD>
<TD ALIGN="middle">$1910</TD>
<TD ALIGN="middle">$1719</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-3E</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-1</STRONG></TD>
<TD ALIGN="middle">$1721</TD>
<TD ALIGN="middle">$1549</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-2</STRONG></TD>
<TD ALIGN="middle">$1721</TD>
<TD ALIGN="middle">$1549</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-3</STRONG></TD>
<TD ALIGN="middle">$1910</TD>
<TD ALIGN="middle">$1719</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-4</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-5</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-6</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
```

```
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-7</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-8</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-9</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
<TD ALIGN="middle">$1893</TD>
</TR>
<TR>
<TD ALIGN="middle"><STRONG>O-10</STRONG></TD>
<TD ALIGN="middle">$2103</TD>
<TD ALIGN="middle">$1893</TD>
</TR>
</TABLE>
<P>
</CENTER>
<H4>Effective date for these allowances: 01-SEP-85</H4>
<H4>The local currency for this area is: Unknown Currency
Name</H4>
<H4>The exchange rate is:  .7042</H4>
<H4>Moving-In Housing Allowance is: $431</H4>
<H4>The climate in this area is: Warm</H4>
<BR>
<HR>
This page was produced by the
<B>Oracle Web Agent</B> on October   25, 1996 09:11 AM<BR>
<A
HREF="/oha/owa/owa_util.showsource?cname=OHA_PKG1.tstoha2">V
iew PL/SQL source code</A>
<HR>
<A HREF="index.html"><IMG
SRC="http://206.39.184.1/oha/earthico.gif"></A>
<P>
<P>
<P>
</BODY>
```

```
</HTML>
```

REFERENCES

1. Blakely, Michael and and Karish, Chuck, "Performance Benchmark Test of the Netcape Fast Track Server", Mindcraft, Inc., 1996 [document on-line]; available from http://www.mindcraft.com/services/web/ns01-fasttrack-nt.html; Internet; accessed on November 18, 1996.

2. Gosling, James and McGilton, Henry, "The Java Language Environment: A White Paper", Sun Microsystems, Inc., 1996.

3. Dwight, Jeffry and Erwin, Michael, *Special Edition Using CGI*, Indianapolis, IN: Que Corporation, 1996.

4. Vowler, Julia, "Why Databases are Getting Wired", *Computer Weekly*, p. 18, Feb. 15, 1996.

5. Lonroth, Magnus, *Oracle Webserver 2.0 Technical Note*, Oracle Corporation, 1996.

6. Gillmor, Dan, "Q&A With Ellison and Lane", *San Jose Mercury News*, p. 5E, November 11, 1996.

BIBLIOGRAPHY

Basiri, Kaveh, "Programming with the NSAPI", Netscape Internet Developer Conference, San Francisco, CA, March 5-7, 1996.

Blakely, Michael and and Karish, Chuck, "Performance Benchmark Test of the Netcape Fast Track Server", Mindcraft, Inc., 1996 [document on-line]; available from http://www.mindcraft.com/services/web/ns01-fasttrack-nt.html; Internet; accessed on November 18, 1996.

Cattell, Rick and Hamilton, Graham, *JDBC: A Java SQL API*, Sun Microsystems, Inc., 1996.

Chen, Frank, Taher, Elgamal, and Treuhaft, Jeff, "Securing Communications on the Intranet and Over the Internet", Netscape Communications Corporation, 1996 [document on-line]; available from http://home.netscape.com/newsref/ref/128bit.html; Internet; accessed on October 15, 1995.

Date, C. J., *An Introduction to Database Systems*, Reading, MA: Addison-Wesley Publishing Company, 1990.

Dwight, Jeffry and Erwin, Michael, *Special Edition Using CGI*, Indianapolis, IN: Que Corporation, 1996.

Gillmor, Dan, "Q&A With Ellison and Lane", *San Jose Mercury News*, p. 1E,5E, November 11, 1996.

Gosling, James and McGilton, Henry, "The Java Language Environment: A White Paper", Sun Microsystems, Inc., 1996.

Gruber, Martin and Rossi, Kennan, *Oracle Web Server User's Guide,* Oracle Corporation, 1996.

"The Internet Application Framework: A White Paper", Netscape Communications Corporation, 1996 [document on-line]; available from http://home.netscape.com/comprod/server_central/tech_docs/oif.html; Internet; accessed on September 9, 1996.

Kim, Pyung-Chul, "A Taxonomy on the Architecture of Database Gateways for the Web", Korea: Chungnam University, July 9, 1996.

Lonroth, Magnus, *Oracle Webserver 2.0 Technical Note*, Oracle Corporation, 1996.

Murdock, Michelle, Price, Mark, and Talley, Brooks, “It’s Between You and Them”, *Infoworld*, p. 70, July 29, 1996.

Rowe, Jeff, *Building Internet Database Servers with CGI*, Indianapolis, IN: New Riders Publishing, 1996.

Shah, Rawn, “Integrating Databases with Java via JDBC”, *JavaWorld*, May 1996 [magazine on-line]; available from http://www.javaworld.com/jw-05-1996/jw-05-shah.html; Internet; accessed on July 25, 1996.

“A Specification for Writing Internet Server Applications”, Microsoft Corporation, 1996 [document on-line]; available from http://www.microsoft.com/win32dev/apiext/isapi.htm; Internet; accessed on August 27, 1996.

Vowler, Julia, ”Why Databases are Getting Wired”, *Computer Weekly*, p. 18, Feb. 15, 1996.

Whetzel, John K., “Integrating the World Wide Web and Database Technology”, *AT&T Technical Journal*, pp. 38-46, March/April 1996.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
   8725 John J. Kingman Road., Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library..2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Per Diem, Travel, and Transportation Allowance Committee......................................1
   Room 836, Hoffman Bldg. 1
   2461 Eisenhower Ave.
   Alexandria, VA 22331-1300

4. Attn: Robert J. Brandewie...1
   Defense Manpower Data Center
   DoD Center, Monterey Bay
   400 Gigling Rd.
   Seaside, CA 93055

5. C. Thomas Wu, Code CS/Wq..2
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943-5101

6. Attn: Deborah Paquette Davis..1
Defense Manpower Data Center
DoD Center, Monterey Bay
400 Gigling Rd.
Seaside, CA 93055

7. Attn: Julie Cornell...2
Defense Manpower Data Center
DoD Center, Monterey Bay
400 Gigling Rd.
Seaside, CA 93055